Espacenet

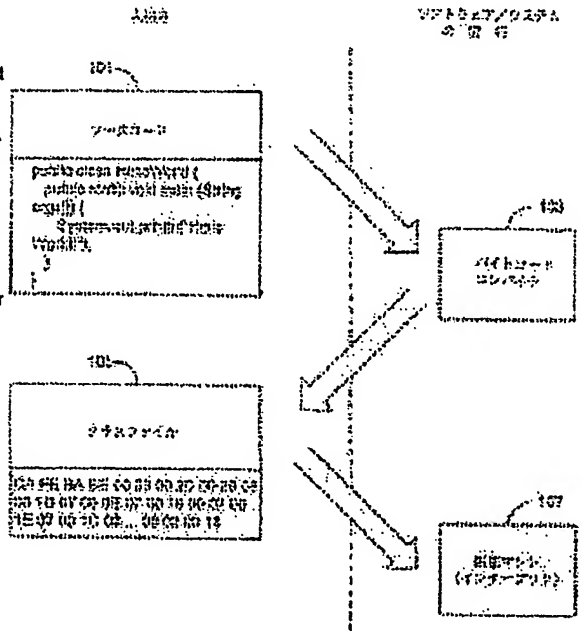
Bibliographic data: JP 11312097 (A)

OBJECT HEAP ANALYSIS TECHNIQUES FOR FINDING OUT MEMORY LEAK AND OTHER RUN TIME INFORMATION

Publication date:	1999-11-09
Inventor(s):	FOOTE WILLIAM F; NISEWANGER JEFFREY D ±
Applicant(s):	SUN MICROSYSTEMS INC ±
Classification:	- international: *G06F11/28; G06F11/36; G06F9/44;* (IPC1-7): G06F11/28; G06F9/44 - European: G06F11/36B5; G06F11/36B9; G06F11/36D3
Application number:	JP19980350354 19981209
Priority number(s):	US19970067993P 19971209; US19980060226 19980414
Also published as:	• EP 0923028 (A2) • EP 0923028 (A3) • EP 0923028 (B1) • US 6167535 (A) • DE 69813160 (T2) • more

Abstract of JP 11312097 (A)

PROBLEM TO BE SOLVED: To provide a snap shot of an object in a run time of an application by preserving the snap shot of an active object at a specified time point during execution. SOLUTION: A source code is inputted to a byte code compiler 103 for compiling that source code. A computer mounting method for analyzing the execution of an object oriented-program includes reception of input during a run time for preserving information concerning an active object. In order to specify the active object, this system can start scanning the active object from an object as the route set member of the object. Next, the object referred to by the route set of the object is identified and the object referred to by these objects is identified.



(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平11－312097

(43)公開日　平成11年(1999)11月９日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　11/28	３１０	Ｇ０６Ｆ　11/28	３１０Ａ
9/44	５３０	9/44	５３０Ｋ

審査請求　未請求　請求項の数１　ＯＬ　外国語出願　（全 69 頁）

(21)出願番号　特願平10－350354

(22)出願日　平成10年(1998)12月９日

(31)優先権主張番号　６０／０６７９９３

(32)優先日　1997年12月９日

(33)優先権主張国　米国（ＵＳ）

(31)優先権主張番号　０９／０６０２２６

(32)優先日　1998年４月14日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　595034134
サン・マイクロシステムズ・インコーポレイテッド
Ｓｕｎ　Ｍｉｃｒｏｓｙｓｔｅｍｓ，　Ｉｎｃ．
アメリカ合衆国　カリフォルニア州　94303　パロ　アルト　サン　アントニオ　ロード　901

(72)発明者　ウィリアム　エフ．　フット
アメリカ合衆国，　カリフォルニア州，　キュパティノ，　ウォルナット　サークル　サウス　22441，　ナンバーピー

(74)代理人　弁理士　長谷川　芳樹　（外２名）

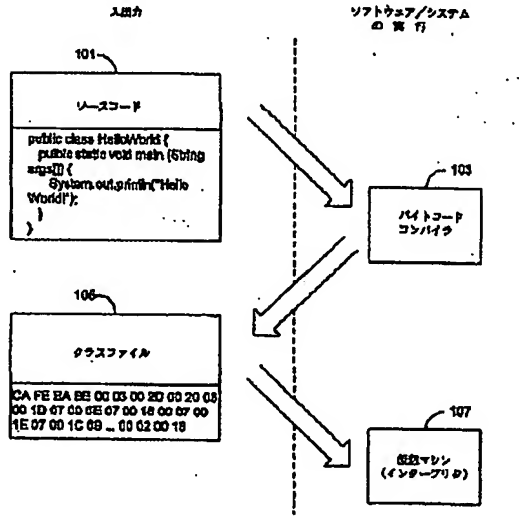
最終頁に続く

(54)【発明の名称】　メモリリーク及び他のランタイム情報を発見するためのオブジェクトヒープ解析技術

(57)【要約】

【課題】　オブジェクト指向コンピュータプログラムを解析するための技術を提供する。

【解決手段】　実行中の特定時点でアクティブであるオブジェクトのスナップショットを保存できる。解析ツールを利用して、ユーザによるアクティブオブジェクトの解析を可能にするハイパーテキストドキュメントを作成できる。更に、ユーザは、２つの異なるランタイムにおけるアクティブオブジェクトの２つの異なるスナップショットを比較でき、その結果、例えば、クラスの新規インスタンスを容易に識別できる。



1

【特許請求の範囲】

【請求項１】コンピュータシステムにおいて、オブジェクト指向プログラムの実行を解析するための方法であって、

アクティブオブジェクトに関する情報を保存するために前記オブジェクト指向プログラムのランタイム中に入力を受取るステップと、

オブジェクトのルートセットのメンバであるオブジェクトから開始して、前記アクティブオブジェクトを求めてスキャンするステップと、

アクティブオブジェクトに関する前記情報を保存するステップと、を有する方法。

【請求項２】アクティブオブジェクトに関する前記情報のスキャンが、オブジェクトのルートセット内のオブジェクトによって参照されるオブジェクトを求めてスキャンするステップを含む、請求項１記載の方法。

【請求項３】アクティブオブジェクトに関する情報の保存は、オブジェクトのルートセット内のオブジェクトと、オブジェクトのルートセット内のオブジェクトによって参照されるオブジェクトとに関する情報を保存するステップを含む、請求項１記載の方法。

【請求項４】アクティブオブジェクトに関する前記情報が、オブジェクトのルートセットのメンバである前記オブジェクトの表示を含む、請求項１又は２の何れか１項記載の方法。

【請求項５】オブジェクトのルートセット内の各オブジェクトは、静的データメンバ、スタックからのローカルリファレンス、又はネイティブコードからのリファレンスによって参照される、請求項１～４の何れか１項記載の方法。

【請求項６】前記アクティブオブジェクトが、他のオブジェクトの構造を定義するオブジェクトを含む、請求項１～５の何れか１項記載の方法。

【請求項７】オブジェクトであるオブジェクトに関する前記情報が、ファイルに保存される、請求項１～６の何れか１項記載の方法。

【請求項８】前記アクティブオブジェクトが、Ｊａｖａクラスのインスタンスを含む、請求項１～７の何れか１項記載の方法。

【請求項９】オブジェクト指向プログラムの実行を解析するためのコンピュータプログラム製品であって、

前記オブジェクト指向プログラムのランタイム中に入力を受取って、アクティブオブジェクトに関する情報を保存するコンピュータコードと、

オブジェクトのルートセットのメンバであるオブジェクトから開始し、前記アクティブオブジェクトを求めてスキャンするコンピュータコードと、

アクティブオブジェクトに関する前記情報を保存するコンピュータコードと、

前記コンピュータコードを保存するコンピュータ読取り

2

可能媒体と、を備える、コンピュータプログラム製品。

【請求項１０】アクティブオブジェクトに関する前記情報を求めてスキャンする前記コンピュータコードが、オブジェクトのルートセット内のオブジェクトによって参照されるオブジェクトを求めてスキャンするコンピュータコードを含む、請求項９記載のコンピュータプログラム製品。

【請求項１１】アクティブオブジェクトに関する前記情報を保存する前記コンピュータコードが、オブジェクトのルートセット内のオブジェクトと、オブジェクトのルートセット内のオブジェクトによって参照されるオブジェクトとに関する情報を保存するコンピュータコードを含む、請求項９又は１０の何れか１項記載のコンピュータプログラム製品。

【請求項１２】アクティブオブジェクトに関する前記情報が、オブジェクトのルートセットのメンバである前記オブジェクトの表示を含む、請求項９～１１の何れか１項記載のコンピュータプログラム製品。

【請求項１３】オブジェクトのルートセット内の各オブジェクトが、静的データメンバ、スタックからのローカルリファレンス、又はネイティブコードからのリファレンスによって参照される、請求項９～１２の何れか１項記載のコンピュータプログラム製品。

【請求項１４】前記アクティブオブジェクトが、他のオブジェクトの構造を定義するオブジェクトを含む、請求項９～１３の何れか１項記載のコンピュータプログラム製品。

【請求項１５】オブジェクトであるオブジェクトに関する前記情報が、ファイルに保存される、請求項９～１４の何れか１項記載のコンピュータプログラム製品。

【請求項１６】前記アクティブオブジェクトが、Ｊａｖａクラスのインスタンスを含む、請求項９～１５の何れか１項記載のコンピュータプログラム製品。

【請求項１７】前記コンピュータ読取り可能媒体が、ＣＤ－ＲＯＭ、フロッピーディスク、テープ、フラッシュメモリ、システムメモリ、ハードドライブ、及び搬送波に組み込まれたデータ信号、から成るグループから選択される、請求項９～１６の何れか１項記載のコンピュータプログラム製品。

【請求項１８】コンピュータシステムにおいて、オブジェクト指向プログラムの実行を解析するための方法であって、

オブジェクト指向プログラムの実行の第１の時点におけるアクティブオブジェクトに関する情報を検索するステップと、

アクティブオブジェクトに関する前記情報を提示するためにハイパーテキストドキュメントを作成するステップと、

前記ハイパーテキストドキュメントを解析のためにユーザに提示するステップと、を有する方法。

【請求項19】更に、オブジェクト指向プログラムの実行の第2の時点におけるアクティブオブジェクトに関する情報を検索するステップを有する、請求項18記載の方法。

【請求項20】前記第1及び第2の時点におけるアクティブオブジェクト間の差分を確定するステップを更に有する、請求項19記載の方法。

【請求項21】前記差分は、前記第2の時点では存在するが、前記第1の時点では存在しないアクティブオブジェクトを含む、請求項20記載の方法。

【請求項22】前記アクティブオブジェクトは、Ｊａｖａクラスのインスタンスを含み、前記ハイパーテキストドキュメントは、HyperText Markup Language（ＨＴＭＬ）で書かれる、請求項18～21の何れか1項記載の方法。

【請求項23】オブジェクト指向プログラムの実行を解析するためのコンピュータプログラム製品であって、
オブジェクト指向プログラムの実行の第1の時点におけるアクティブオブジェクトに関する情報を検索するコンピュータコードと、
アクティブオブジェクトに関する前記情報を提示するためにハイパーテキストドキュメントを作成するコンピュータコードと、
前記ハイパーテキストドキュメントを解析のためにユーザに提示するコンピュータコードと、
前記コンピュータコードを保存するコンピュータ読取り可能媒体と、を備える、コンピュータプログラム製品。

【請求項24】　オブジェクト指向プログラムの実行の第2の時点におけるアクティブオブジェクトに関する情報を検索するコンピュータコードと、
前記第1及び第2の時点におけるアクティブオブジェクト間の差分を決定するコンピュータコードであって、前記差分は、前記第2時点では存在するが、前記第1時点では存在しないアクティブオブジェクトを含むようにしたコンピュータコードと、を更に備える、請求項23記載のコンピュータプログラム製品。

【請求項25】コンピュータシステムにおいて、オブジェクト指向プログラムの実行を解析するための方法であって、
オブジェクト指向プログラムのランタイム中に、オブジェクト指向プログラムの実行の第1及び第2の時点におけるアクティブオブジェクトに関する情報を保存するステップと、
オブジェクト指向プログラムの実行の前記第1及び第2の時点におけるアクティブオブジェクトに関する情報を検索するステップと、
前記第1及び第2の時点におけるアクティブオブジェクト間の差分を確定するステップと、
前記第1及び第2の時点におけるアクティブオブジェクト間の前記差分を提示するためにハイパーテキストドキュメントを作成するステップと、

解析のためにユーザに対して前記第1及び第2の時点におけるアクティブオブジェクト間の前記差分を提示するために前記ハイパーテキストドキュメントを提示するステップと、を有する方法。

【請求項26】オブジェクト指向プログラムの実行を解析するためのコンピュータプログラム製品であって、
オブジェクト指向プログラムのランタイム中に、オブジェクト指向プログラムの実行の第1及び第2の時点におけるアクティブオブジェクトに関する情報を保存するコンピュータコードと、
オブジェクト指向プログラムの実行の前記第1及び第2の時点におけるアクティブオブジェクトに関する情報を検索するコンピュータコードと、
前記第1及び第2の時点におけるアクティブオブジェクト間の差分を確定するコンピュータコードと、
前記第1及び第2の時点におけるアクティブオブジェクト間の前記差分を提示するためにハイパーテキストドキュメントを作成するコンピュータコードと、
解析のためにユーザに対して前記第1及び第2の時点におけるアクティブオブジェクト間の前記差分を提示するために前記ハイパーテキストドキュメントを提示するコンピュータコードと、
前記コンピュータコードを保存するコンピュータ読取り可能媒体と、を備える、コンピュータプログラム製品。

【発明の詳細な説明】

【０００１】

【発明の背景】本発明はソフトウェアアプリケーションを解析（例えば、最適化、エラーの特定、又は「デバッグ」）するための技術に関する。より詳細には、本発明は、その後の解析のために、Ｊａｖａ（商標）仮想マシン上のアプリケーションのランタイム中に存在するオブジェクトに関する情報を保存するための技術に関する。

【０００２】Ｊａｖａ（商標）プログラミング言語は、Sun Microsystemsによって開発され、小型デバイス（例えば、ポケベル、携帯電話、及びスマートカード）からスーパーコンピュータまでの広範囲のコンピュータ上で実行できるように充分ポータブルに設計されたオブジェクト指向高級プログラミング言語である。Ｊａｖａ（及びその他言語）で書かれたコンピュータプログラムは、Ｊａｖａ仮想マシンによる実行のための仮想マシン命令にコンパイルできる。一般に、Ｊａｖａ仮想マシンは、仮想マシン命令を解読、実行するインタプリタである。

【０００３】Ｊａｖａ仮想マシン用の仮想マシン命令はバイトコードであり、それらがひとつ以上のバイトを含むことを意味する。バイトコードは、ひとつのクラスのメソッド用バイトコードを含む「クラスファイル」と呼ばれる特定のファイルフォーマットに保存される。クラスのメソッド用バイトコードに加えて、クラスファイルは、記号テーブルならびに他の補助的情報を含む。

【０００４】ひとつ以上のクラスファイル中のＪａｖａバイトコードとして具体化されるコンピュータプログラムは、プラットフォームに依存しない。そのコンピュータプログラムは、Ｊａｖａ仮想マシンの実装を、実行可能な任意のコンピュータ上で実行し、アンモディファイ(unmodified)できる。Ｊａｖａ仮想マシンは、Ｊａｖａ仮想マシン用コンピュータプログラムをプラットフォームに依存しないようにするときの主要ファクタである「一般的(generic)」コンピュータのソフトウェアエミュレータである。

【０００５】Ｊａｖａ仮想マシンは、ソフトウェアインタプリタとして通常的に実装されている。従来のインタプリタは、解釈されたプログラムの仮想マシン命令を、実行中に一度に一命令ずつ解読、実行するが、これは、実行に先立ってソースコードをネイティブマシン命令に解読するので、実行中に解読を行なわないコンパイラとは対照的である。通常、Ｊａｖａ仮想マシンは、Ｊａｖａプログラミング言語以外のプログラミング言語（例えば、Ｃ＋＋プログラミング言語）で書かれる。従って、Ｊａｖａプログラムの実行は、多数のプログラミング言語で書かれた関数の実行を伴うことができる。更に、バイトコード自体が、Ｊａｖａプログラミング言語で書かれていない関数（例えば、入出力用のシステム関数）を呼び出すことができる。従って、Ｊａｖａプログラムの実行は、多数のプログラミング言語で書かれた関数の実行を伴うのが普通である。

【０００６】テストされたソースコードの再利用と、その結果のランタイムエラーの数の低減を可能にすることが、オブジェクト指向プログラムの目標だが、Ｊａｖａプログラムは依然として、ランタイム中にプログラムの操作に対してウインドウを提供する解析技術から利益を得ることができる。例えば、その解析を利用してプログラムを最適化したり、コード中のバグを特定することができる。従って、Ｊａｖａ仮想マシン上で実行するアプリケーションを解析する革新的技術を提供することが望ましいだろう。更に、例えば、メモリリークを検出できるように、アプリケーションのランタイム中に存在するオブジェクトのスナップショットを提供することは利益となるだろう。

【０００７】

【発明の概要】一般に、本発明の実施形態は、オブジェクト指向コンピュータプログラムを解析するための革新的技術を提供する。実行中の特定時点におけるアクティブなオブジェクト（「アクティブオブジェクト」）のスナップショットを保存できる。解析ツールを利用して、ユーザによるアクティブオブジェクトの解析を可能にするハイパーテキストドキュメントを作成できる。例として、ユーザは、HyperText Markup Language（HTML）によってドキュメントをブラウズして、クラスのインスタンスが予想よりも多すぎることを確定することによって、メモリリークを確認できるであろう。ユーザは次に、何がメモリリークを発生させているかを確定するために、不要なインスタンスへのリファレンス（又はポインタ）を追跡できる。更に、ユーザは、２つの異なるランタイムにおけるアクティブオブジェクトの２つの異なるスナップショットを比較して、その結果、例えば、クラスの新規インスタンスを容易に識別できる。本発明のいくつかの実施形態を以下に説明する。

【０００８】一実施形態では、オブジェクト指向プログラムの実行を解析するためのコンピュータ実装メソッドが、アクティブオブジェクトに関する情報を保存するためにランタイム中に入力を受取ることを含む。アクティブオブジェクトを特定するために、システムは、オブジェクトのルートセットのメンバであるオブジェクトから始めて、アクティブオブジェクトのスキャンを開始できる。次に、オブジェクトのルートセットによって参照(referenced)されたオブジェクトが識別され、これらのオブジェクトによって参照されたオブジェクトが識別される等と、すべてのアクティブオブジェクトが識別されるまで続く。アクティブオブジェクトに関する情報は、（例えば、ファイル内に）保存することができる。好ましい実施形態では、アクティブオブジェクトは、Ｊａｖａクラスのインスタンスである。

【０００９】別の実施形態では、オブジェクト指向プログラムの実行を解析するためのコンピュータ実装メソッドが、実行の或る時点におけるアクティブオブジェクトに関する情報を検索することを含む。情報を繰り返しスキャンすることによって、ルートセットのメンバであってそのアクティブオブジェクトの階層を維持するオブジェクトを識別できる。ハイパーテキストドキュメントを作成して、アクティブオブジェクトに関する情報を解析のためにユーザに提示してもよい。

【００１０】別の実施形態では、オブジェクト指向プログラムの実行を解析するためのコンピュータ実装メソッドが、２つの異なるランタイムにおけるアクティブオブジェクトに関する情報を保存することを含む。第１と第２のランタイム間の差分、例えば新規インスタンス、を次に決定できる。ハイパーテキストドキュメントを作成して、その差分を、解析のためにユーザに提示できる。

【００１１】本発明の他の特徴と利点は、添付の図面に関連して下記の詳細な説明を一読することによって直ちに明らかになるであろう。

【００１２】

【好ましい実施形態の詳細な説明】定義

クラス　－　同一特性を共有するオブジェクトを定義したオブジェクト指向データ型で、通常、データとそのデータ上で動作する関数の両方を含む。

【００１３】オブジェクト（又はインスタンス）　－　クラスの例示されたメンバ。

【００１４】オブジェクトのルートセット　－　他のオ

プジェクトを介して連鎖することなく、直接に参照可能なオブジェクト。

【0015】アクティブオブジェクト　－　例示されたオブジェクトであって、ルートセットのメンバであるか、直接又は間接にルートセットのメンバによって参照可能なオブジェクト。

【0016】ネイティブメソッド（又はコード）　－　Ｊａｖａプログラミング言語以外のプログラミング言語で書かれた関数。

概要

以下の説明では、本発明を、実行中に、Ｊａｖａプログラム（例えば、バイトコード）の実行を解析する好ましい実施形態に関して説明する。特に、Ｊａｖａ仮想マシンがＣ＋＋プログラミング言語で書かれた例について説明する。しかしながら、本発明は、特定の言語、コンピュータアーキテクチャ、又は特定の実装に限定されない。従って、以下の実施形態の記述は説明のためであって、限定のためではない。

【0017】図１は、本発明の実施形態のソフトウェアを実行するために使用可能なコンピュータシステムの例を示す。図１は、ディスプレイ３、スクリーン５、キャビネット７、キーボード９、及びマウス１１を含むコンピュータシステム１を示す。マウス１１は、グラフィカルユーザインタフェースと対話するためのひとつ以上のボタンを持つことができる。キャビネット７は、本発明を実行するコンピュータコードを組み込んだソフトウェアプログラムや本発明と共に使用されるデータ等を保存、検索するために利用可能なＣＤ－ＲＯＭドライブ１３、システムメモリ、及びハードドライブ（図２参照）を内蔵している。代表的なコンピュータ読取り可能記憶媒体としてＣＤ－ＲＯＭ１５を示すが、フロッピーディスク、テープ、フラッシュメモリ、システムメモリ、及びハードドライブを含む他のコンピュータ読取り可能記憶媒体も利用できる。更に、搬送波（例えば、インタネットを含むネットワーク）に組み込まれたデータ信号が、コンピュータ読取り可能記憶媒体であってもよい。

【0018】図２は、本発明の実施形態のソフトウェアを実行するために使用されるコンピュータシステム１のシステムブロック図を示す。図１に示すように、コンピュータシステム１はモニタ３とキーボード９、及びマウス１１を含む。コンピュータシステム１は更に、中央プロセッサ５１、システムメモリ５３、固定記憶装置５５（例えば、ハードドライブ）、リムーバブル記憶装置５７（例えば、ＣＤ－ＲＯＭドライブ）、ディスプレイアダプタ５９、サウンドカード６１、スピーカ６３、及びネットワークインタフェース６５等のサブシステムを含む。本発明での使用に適した他のコンピュータシステムは、追加又はより少数のサブシステムを含んでもよい。例えば、別のコンピュータシステムは、２つ以上のプロセッサ５１（すなわち、マルチプロセッサシステム）やキャッシュメモリを含むことができるだろう。

【0019】コンピュータシステム１のシステムバスアーキテクチャを矢印６７で表わす。しかしながら、これらの矢印は、サブシステムをリンクするのに役立つ任意の相互接続構造を示している。例えば、ローカルバスを利用して、中央プロセッサをシステムメモリとディスプレイアダプタに接続できるであろう。しかし、図２に示すコンピュータシステム１は、本発明での使用に適したコンピュータシステムの一例に過ぎない。異なる構成のサブシステムを持つ他のコンピュータアーキテクチャを利用することもできる。

【0020】通常、Ｊａｖａプログラミング言語で書かれたコンピュータプログラムは、その後にＪａｖａ仮想マシンによって実行されるバイトコードかＪａｖａ仮想マシン命令にコンパイルされる。バイトコードは、解釈のためにＪａｖａ仮想マシンに入力されるクラスファイルに保存される。図３は、Ｊａｖａ仮想マシンであるインタプリタによる実行を介してＪａｖａソースコードの簡単なピースの進行を示す。

【0021】Ｊａｖａソースコード１０１は、Ｊａｖａで書かれた古典的なHello Worldプログラムを含む。ソースコードは次に、そのソースコードをバイトコードにコンパイルするバイトコードコンパイラ１０３に入力される。バイトコードは、それらがソフトウェアをエミュレーションしたコンピュータによって実行されるので、仮想マシン命令である。通常、仮想マシン命令は、一般的（すなわち、特定のマイクロプロセッサやコンピュータアーキテクチャ用に設計されていない）だが、これは必要ではない。バイトコードコンパイラは、Ｊａｖａプログラム用のバイトコードを含むＪａｖａクラスファイル１０５を出力する。

【0022】Ｊａｖａクラスファイルは、Ｊａｖａ仮想マシン１０７に入力される。Ｊａｖａ仮想マシンは、Ｊａｖａクラスファイル中のバイトコードを解読、実行するインタプリタである。Ｊａｖａ仮想マシンはインタプリタだが、一般には、ソフトウェア内でマイクロプロセッサやコンピュータアーキテクチャ（例えばハードウェア中に存在しないマイクロプロセッサやコンピュータアーキテクチャ）をエミュレートするような仮想マシンと呼ばれている。

【0023】図４はＪａｖａランタイムシステムの実装のコンポーネントを示す。Ｊａｖａ仮想マシンの実装は、Ｊａｖａランタイムシステムとして知られている。Ｊａｖａランタイムシステム２０１は、Ｊａｖａプログラムを実行するために、Ｊａｖａクラスファイル２０３、標準ビルトインＪａｖａクラス２０５、及びネイティブメソッド２０７の入力を受取る。標準ビルトインＪａｖａクラスは、スレッド、ストリング等のオブジェクト用のクラスでよい。ネイティブメソッドはＪａｖａプログラミング言語以外のプログラミング言語で書いても

よい。ネイティブメソッドは通常、ダイナミックリンクライブラリ（ＤＬＬｓ）か共有ライブラリに保存されている。

【００２４】Ｊａｖａランタイムシステムは、オペレーティングシステム２０９とインタフェースすることもできる。例えば入出力関数は、Ｊａｖａクラスファイル２０３、標準ビルトインＪａｖａクラス２０５、及びネイティブメソッド２０７へのインタフェースをＪａｖａランタイムシステムに備えることを含めて、オペレーティングシステムによって処理してもよい。

【００２５】動的クラスローダ及びベリファイヤ２１１は、Ｊａｖａクラスファイル２０３と標準ビルトインＪａｖａクラス２０５を、オペレーティングシステム２０９を介してメモリ２１３にロードする。更に、動的クラスローダ及びベリファイヤは、Ｊａｖａクラスファイル中のバイトコードの正しさを確認して、検出した何れのエラーも報告できる。

【００２６】ネイティブメソッドリンカ２１５は、ネイティブメソッド２０７をオペレーティングシステム２０９を介してＪａｖａランタイムシステムにリンクさせ、ネイティブメソッドをメモリ２１３に保存する。図示のように、メモリ２１３はＪａｖａクラス用のクラスとメソッド領域、及びネイティブメソッド用のネイティブメソッド領域を含んでもよい。メモリ２１３のクラスとメソッド領域は、ガベージコレクションされたヒープに保存できる。新規オブジェクトが作成されると、それらはガベージコレクションされたヒープに保存される。アプリケーションではなく、Ｊａｖａランタイムシステムが、スペースが最早利用されていないときに、ガベージコレクションされたヒープのメモリを再利用(reclaim)する責任がある。

【００２７】図４に示すＪａｖａランタイムシステムの心臓部には、実行エンジン２１７がある。実行エンジンはメモリ２１３に保存された命令を実行すると共に、ソフトウェア、ハードウェア、又は両者の組合せの形で実装できる。実行エンジンはオブジェクト指向アプリケーションをサポートし、概念的には各Ｊａｖａスレッド毎にひとつずつの同時に実行する多数の実行エンジンが存在する。実行エンジン２１７はまた、サポートコード２２１も利用できる。サポートコードは、例外、スレッド、セキュリティ等に関連する機能性を提供できる。

【００２８】Ｊａｖａプログラムの実行に従って、関数が各スレッド内で逐次的に呼び出される。各スレッド毎に、実行を終えていない各関数用のフレームを保存する実行スタックが存在する。フレームは関数の実行のための情報を保存するが、その情報は状態変数、ローカル変数、及びオペランドスタックを含んでもよい。関数が呼び出されると、その関数用のフレームを実行スタックにプッシュする。関数が終了すると、その関数のフレームを実行スタックからポップする。従って、実行スタックの頂部のフレームに対応する関数のみがアクティブで、実行スタックの頂部より下のフレームに対応する関数は、それらが呼び出した関数が戻る（すなわち、終了する）まで、それらの実行が中断されている。

【００２９】図５は、実行スタック上に保存されているＪａｖａ関数用フレームを示す。実行スタックの頂部のフレーム３０３と、フレーム３０３の下に保存されたフレーム３０５と３０７とをそれぞれ有する実行スタック３０１が示されている。スタックポインタＳＰは実行スタックの頂部を指すが、フレームポインタＦＰは、実行スタック３０１の頂部のフレーム内のフレームポインタを指す。

【００３０】各フレームは、そのフレームに対応する関数用の状態変数、ローカル変数、及びオペランドを含むように示されている。更に、フレームに保存された最後の項目は、矢印３０９、３１１によって示されるような、実行スタック上のカレントフレームの下のフレームのフレームポインタを指すフレームポインタである。ネイティブメソッド用のフレームを保存する実行スタックは、異なっているように見えるかもしれないが、基本原理は通常、図示の実行スタックと同様である。

ランタイム実行の解析

Ｊａｖａプログラムの実行を解析するために、ユーザはプログラムをＪａｖａ仮想マシンで実行する。仮想マシンはＪａｖａプログラムを解釈する責任があり、効率の増大のためにコンパイルを行なってもよい。図６は、Ｊａｖａプログラムのランタイム中に得られるアクティブオブジェクトに関する情報を保存するプロセスを示す。

【００３１】ステップ４０１で、システムは、アクティブオブジェクトに関する情報を保存するためにランタイム中に入力を受取る。例えば、ユーザは一定のキーストローク（例えば、<control>＼）を押して、アクティブオブジェクトに関する情報を保存すべき旨をシステムに指示してもよい。

【００３２】システムは次に、ステップ４０３でルートセットのメンバから始めて、要求された情報を求めてスキャンする。ルートセットのメンバであるオブジェクトを、直接に参照してもよい。例えば、好ましい実施形態では、ルートセットのメンバであるオブジェクトをＪａｖａクラスの静的データメンバによって参照してもよいし、Ｊａｖａ又はネイティブ実行スタックによってローカルに参照してもよいし、ネイティブコード（グローバルＪａｖａネイティブインタフェースリファレンス）によってグローバルに参照してもよい。システムは、ルートセットのメンバであるオブジェクトで開始し、ルートセットのメンバによって参照された他の任意のオブジェクトに対するリファレンスを繰り返し追跡してもよい。かくして、ルートセットのメンバであるか、ルートセットのメンバから直接又は間接に参照され得るすべてのオブジェクトが識別される（すなわち、ルートセットから

到達可能なオブジェクトの過渡クロージャ(transitive closure))。これらのオブジェクトはこの時点でアクティブオブジェクトである。好ましい実施形態では、マークアンドスイープアルゴリズム(mark and sweep algorithm)がガベージコレクションアルゴリズム(garbage collection algorithm)と同様に利用される。

【００３３】アクティブオブジェクトが識別されると、システムはステップ４０５で、そのオブジェクトに関する情報を保存する。情報は、オブジェクトのｉｄ、オブジェクトがルートセットのメンバであるか否か、及びその理由（例えば、Ｊａｖａクラスの静的データメンバによって参照されたのか）を示すフラグ、該当する場合はスレッドｉｄ、クラスｉｄ、オブジェクトのデータ等を含んでもよい。好ましい実施形態で保存される情報を図２７Ａ～２７Ｂに示す。Ｊａｖａプログラムでは、アクティブオブジェクトに関する情報が他のオブジェクトの構造を定義するオブジェクトを含むように、クラスClassのオブジェクトが他のオブジェクトの構造（すなわち、特定クラスのすべてのオブジェクトの構造）を保存する。

【００３４】アクティブオブジェクトに関する情報は、ファイル又は任意の他のコンピュータ記憶装置（例えば、メモリ）に保存できる。情報は、ひとつのアクティブオブジェクトが識別されたときに保存してもよいし、その後で、すべて一度に保存してもよい。従って、フローチャートのステップは説明のために図示されたもので、当該技術に精通する者であれば、他の実施形態ではステップを組み合せたり、削除したり、挿入できることを容易に理解するであろう。情報はランタイム中に解析できるが、通常はプログラムが実行を終了した後で解析される。図７は、ランタイム中に得られたアクティブオブジェクトに関する情報を提示するプロセスを示す。ステップ４５１で、システムはアクティブオブジェクトに関する保存情報を検索する。情報はアクティブオブジェクト自体に関する情報ばかりでなく、それらの相互の関係を含んでもよい。アクティブオブジェクト間の関係は、ルートセットのメンバから開始し、通常はインディレクション(indirection) の様々なレベルを通って他のオブジェクトに進む指示グラフ構造(directed graph structure)に似ているであろう。かくして、ルートセットのメンバでないアクティブオブジェクトは、図６のステップ４０３と同様のメソッドでルートセットのメンバからのリファレンスを追跡することによって識別できる。

【００３５】アクティブオブジェクトに関する情報が検索されると、システムはステップ４５３でユーザの問い合せを受ける。好ましい実施形態では、問い合せは、HyperText Transport Protocol（ＨＴＴＰ）サーバーへのUniform Resource Locator（ＵＲＬ）として指定される。利用可能な問い合せについて下記に説明する。

【００３６】「全クラスの問い合せ」は、ランタイムにヒープ上に存在したすべてのクラスを示す。クラスは、それらの完全に承認された(fully-qualified)クラスネームによって分類すると共に、パッケージによって編成することができる。この問い合せの結果の例を図８～１２に示す。

【００３７】「クラスの問い合せ」は、所望のクラスに関する情報を示す。情報はスーパークラス、任意のサブクラス、インスタンスデータメンバ、及び静的データメンバを含んでもよい。この問い合せの結果の例を図１３に示す。

【００３８】「インスタンスの問い合せ」は、指定のクラスのすべてのインスタンスを示す。この問い合せの結果の例を図１４に示す。

【００３９】「オブジェクトの問い合せ」は、ランタイムにヒープ上に存在したオブジェクトに関する情報を示す。最も注目すべきことは、誰でもこのオブジェクトに関係するオブジェクトまでナビゲートできることで、これを利用してエラーを突き止めることができる。この問い合せの結果の例を図１６に示す。

【００４０】「ルートの問い合せ」は、ルートセットから特定オブジェクトまでのリファレンスチェーンを提供する。チェーンは、問題のオブジェクトに到達可能なルートセットの各メンバから提供される。好ましい実施形態では、チェーンは、その長さを低減するために深層第１の検索(depth-first search)によって計算される。他の検索技術も利用できる。「ルートの問い合せ」は、それを利用してオブジェクトがまだアクティブな理由を決定できるので、メモリリークを突き止めるために非常に貴重な問い合せである。この問い合せの結果の例を図１７に示す。

【００４１】「到達可能なオブジェクトの問い合せ」は、特定オブジェクトから到達可能なすべてのオブジェクトの過渡クロージャを示す。この問い合せは、メモリ内のオブジェクトの全ランタイムの足跡を確定するために有用である。この問い合せの結果の例を図（図１６の後）に示す。

【００４２】「全ルートの問い合せ」は、ルートセットのすべてのメンバを示す。この問い合せの結果の例を図？？？に示す。

【００４３】問い合せが、ユーザはステップ４５５で終了した旨、示さない場合は、システムは、ステップ４５５における問い合せを満たす情報を提示するために、ハイパーテキストドキュメント（単数又は複数）を作成する。ハイパーテキストドキュメントは、そのドキュメントの他の部分又は他のドキュメントすべてに対するリンクを含む。好ましい実施形態では、ハイパーテキストドキュメント（単数又は複数）はHyperText Markup Language （ＨＴＭＬ）で書かれている。

【００４４】ステップ４５９で、システムはハイパーテキストドキュメントを解析のためにユーザに提示する。

これは、Ｗｅｂブラウザによって見ることができるＨＴＭＬドキュメントを提示するためのＨＴＴＰサーバーを作成することによって行なわれる。

【００４５】上記は本発明の実施形態を説明したが、解析のためにユーザに提示可能なデータの実例を見ることは読者にとって役立つだろう。アクティブオブジェクトに関する情報のスナップショットが取られてＨＴＭＬドキュメントに挿入されると、ユーザはＷｅｂブラウザを利用してその情報を解析できる。良い出発点は、アクティブだったインスタンスを持ったすべてのケースを示すための問い合せであろう。

【００４６】図８～１２は、スナップショットを取ったときにアクティブだったインスタンスを持ったすべてのクラスのハイパーテキストドキュメントを示す。図示のように、クラスはパッケージによって編成される。アンダーラインの付いた語は、他のＨＴＭＬドキュメントへのハイパーテキストリンクを表す。一例として、図１１のリンク５０３を選択することによってクラスjovial.slotCar.track.TrackSegmentに関する更なる詳細を見ることができる。

【００４７】図１３は、スナップショットを取ったときにアクティブだったインスタンス持ったクラスのハイパーテキストドキュメントを示すが、それはこの場合、クラスjovial.slotCar.track.TrackSegmentである。かくして、ユーザがリンク５０３で「クリック」すれば、ユーザは、図１３に示すハイパーテキストドキュメントを見るだろう。図示のように、スーパークラス、サブクラス、インスタンスデータメンバ、及び静的データメンバ等の情報が、より詳細な情報へのハイパーテキストリンクによって提示できる。

【００４８】図１３のハイパーテキストドキュメントの底部で、ユーザは、サブクラスを除くリンク５５３かサブクラスを含むリンク５７５によって、クラスのアクティブインスタンスを見るように要求できる。言い換えれば、リンク５５３は、クラスjovial.slotCar.track.TrackSegmentのすべてのインスタンスを示すであろうが、このクラスのサブクラス（例えば、クラスjovial.slotCar.track.CurvedTrackSegmentと、jovial.slotCar.track.StraightTrackSegment）のインスタンスを含まない。リンク５７５は、このクラスのサブクラスのインスタンスを含む、クラスjovial.slotCar.track.TrackSegmentのすべてのインスタンスを示すだろう。

【００４９】図１４は、サブクラスを除いたクラスのインスタンスのハイパーテキストドキュメントを示す。そのハイパーテキストドキュメントは、リンク５５３を選択したときに提示されるものではないだろうが、にもかかわらず、それはハイパーテキストドキュメントがどのように見えるかの例を示す。図示のように、指定のメモリロケーションに位置付けされた、クラスjovial.slotCar.Carの２つのインスタンスが存在する。ユーザがこれらのインスタンスのいずれかに関する更なる情報を解析したい場合は、リンクを選択すればよい。リンク６０７を図１６に関して更に詳しく説明する。

【００５０】図１５は、サブクラスを含むクラスのインスタンスのハイパーテキストドキュメントを示す。この図は、ユーザが図１３のリンク５７５をクリックした場合に提示されるものを示す。サブクラスのインスタンスが示されているため、そのインスタンスの一部はjovial.slotCar.track.TrackSegmentよりも多くのデータ又は関数を含むであろう。しかしながら、これらのインスタンスはすべて、この親クラスから継承される。

【００５１】図１６は、アクティブオブジェクトに関する情報のハイパーテキストドキュメントを示す。アクティブオブジェクトは、図１４のリンク６０７によって選択可能なオブジェクトである。図示のように、クラス、インスタンスデータメンバ、オブジェクトへのリファレンス、及びルートセットからのリファレンスチェーンを提示できる。例として、リンク６３５は、弱いリファレンスを除くルートセットからのリファレンスチェーンを示し、リンク６７５は、弱いリファレンスを含むルートセットからのリファレンスチェーンを示すだろう。弱いリファレンスとは、より多くのメモリ又はヒープスペースが必要な場合にガベージコレクタが除去可能なリファレンスである。リンク６８１は、このオブジェクトによって到達可能な（又は、参照される）すべてのオブジェクトを示すだろう。

【００５２】図１７は、弱いリファレンスを除くアクティブオブジェクトへのルートセットリファレンスのハイパーテキストドキュメントを示すが、図１８は、弱いリファレンスを含むアクティブオブジェクトへのルートセットリファレンスのハイパーテキストドキュメントを示す。弱いリファレンスは図示されないが、図１７のリンク６９１はユーザによる図１８への切換えを許すのに対して、図１８のリンク６９３は、ユーザによる戻りを許すだろう。

【００５３】図１９と図２０は、アクティブオブジェクトから到達可能なすべてのオブジェクトのハイパーテキストドキュメントを示す。これは、ユーザが図１６のリンク６８１をアクティブにしたときに表示できる。オブジェクトは、アクティブオブジェクトの指示グラフを検索することによって識別できる。

【００５４】再び図１２を参照すると、リンク７０５によって、ユーザはルートセットのすべてのメンバを見ることを要求できる。図２１～２５は、リンク７０５をアクティブにした場合に提示できるルートセットのすべてのメンバのハイパーテキストドキュメントを示す。図示のように、ルートセットのメンバは、そのメンバをリファレンスする方式に従って編成できる。例えば、Ｊａｖａ静的リファレンス、Ｊａｖａローカルリファレンス、ネイティブ静的リファレンス、及びネイティブローカル

リファレンスによる。図２５のリンク７１５によって、ユーザは、図８～１２のような、アクティブインスタンスを持ったすべてのクラスを要求できる。

【００５５】ハイパーテキストドキュメントの例は、ユーザが好ましい実施形態の中でアクティブオブジェクトを容易に解析できることを説明する。しかしながら、本発明はハイパーテキストドキュメントによる提示に限定されるものではなく、当該技術に精通する者であれば他のメソッドが利用できることを理解するであろう。

【００５６】本発明の更なる理解のために、ユーザがＪａｖａプログラムの解析を行なうようなシナリオの記載が役立つだろう。メモリリークは、殆どすべてのプログラミング言語で普遍的な問題である。メモリリークは一般に、割り当てられてはいるが最早コンピュータプログラムによって使用されていないメモリに関係する。最悪の場合のシナリオでは、利用可能なメモリが充分に存在しないので、メモリリークがメモリアロケーションエラーをもたらす。メモリリークは、最早必要でないヒープ内のオブジェクトへのルートセットからのリファレンスが存在するときの、プログラマーエラーに起因する場合もある。

【００５７】ユーザはメモリリーク問題が存在すると疑っていると仮定しよう。ユーザは、或るランタイムでアクティブでなければならないクラスのインスタンス数を勘定することができる。ユーザは次に、本発明の実施形態に従ってアクティブオブジェクトのスナップショットを作成できる。ユーザは次に、いくつのインスタンスがアクティブかを確定するために、保存情報を解析できる。予想以上に存在する場合、ユーザは、どのオブジェクトが誤ったリファレンスを保持しているかを確定するために、誤ったアクティブオブジェクトからのリファレンスを追跡することができる。

【００５８】更に、本発明の実施形態によって、ユーザはアクティブオブジェクトのスナップショットを比較することができる。図２６は、２つの異なるランタイムにおけるアクティブオブジェクト間の差分（例えば、新規インスタンス）を確定するプロセスを示す。

【００５９】ステップ７５１で、システムは第１ランタイムにおけるアクティブオブジェクトに関する情報を保存する。続いてステップ７５３で、システムは第２ランタイムにおけるアクティブオブジェクトに関する情報を保存する。上記のように、ユーザ入力によって情報を保存するようにシステムに指示できる。

【００６０】ユーザはステップ７５５で、第１と第２ランタイム間の差分を確定するようにシステムに指示してもよい。例えば、ユーザは、２つのランタイムにおけるスナップショットを保存するファイルを指定できる。システムは次に、そのスナップショットを比較して、スナップショット間の差分、例えば如何なる新規インスタンスが第２ランタイムにおいて存在するか、を確定できる。

【００６１】第１と第２ランタイム間の差分が確定されると、システムはステップ７５７でユーザの問い合せを受ける。上記の問い合せに加えて、第２ランタイムでの新規インスタンスのみを示す「新規インスタンスの問い合せ」を利用できる。インスタンスは、それが第１スナップショットには存在するが、第２スナップショットには同一ｉｄを持つオブジェクトが存在しない場合、「新規」と見做すことができる。オブジェクトのｉｄは、仮想マシンによって割り当てられると共にそのオブジェクトを独特のメソッドで識別する３２ビットの整数（又はハンドル）でもよい。ハンドルは再使用できるが、比較的短い時間間隔で取られたスナップショットが、一般に優れた結果を生む。この問い合せの結果の例を図２７に示す。

【００６２】問い合せが、ユーザはステップ７５９で終了したことを示さない場合、システムはステップ７６１で、その問い合せを満たす情報（例えば、２つのスナップショット間の差分）を提示するためにハイパーテキストドキュメント（単数又は複数）を作成する。システムは、ステップ７６３で、通常、Ｗｅｂブラウザで見ることができるＨＴＭＬドキュメントを提示するためのＨＴＴＰサーバーを作成することによって、そのハイパーテキストドキュメントを解析するためにユーザに提示できる。

【００６３】クラスの新規インスタンスのハイパーテキストドキュメントを図２７に示す。図示のように、リンク８０１によって、ユーザはクラスに関する更なる詳細を見ることができるし、リンク８０３によって、ユーザは新規インスタンスに関する更なる詳細を見ることができる。図２７のハイパーテキストドキュメントでは、サブクラスは図示されない。図２８は、サブクラスを含むクラスの新規インスタンスのハイパーテキストドキュメントを示すが、この例では追加のインスタンスは存在しない。

【００６４】デバッグ中、コンピュータプログラムが時間と共にどのように変化するかを解析することは有益であろう。本発明の実施形態では、ユーザは、ランタイム中にアクティブオブジェクトの多数のスナップショットを取れるばかりでなく、システムにスナップショットを比較させて更なる解析のために差分を提示させることができる。

【００６５】この点に対して、アクティブオブジェクトに関する情報がどのように保存されるかについて本明細書は詳しく説明する。図２９～３０は、好ましい実施形態でアクティブオブジェクトに関する情報の保存のために利用可能なデータ構造を説明する。これらの図に明示される構造に従って、バイナリオブジェクトダンプ（ＢＯＤ）ファイルを保存してもよい。この構造は好ましい実施形態で利用可能だが、他の構造も本発明の精神から

逸脱することなく利用できる。

【００６６】図３１はＢＯＤサーバーのモデルパートを示す。サーバーが起動されると、それがＢＯＤファイルを読み取り、メモリに示される構造を構築する。この構造の頂部レベルはSnapshotのインスタンスである（図示のオブジェクトはSnapshotのデータメンバである）。ユーザが情報を要求すると、サーバーはスナップショットを、ヒープの内容の読取り専用の表示として扱う。

結論

上記は本発明の好ましい実施形態の完全な説明だが、代案や修正案や同等仕様も使用可能である。本発明は、上記の実施形態を適切に変更することによって同様に適用できることは言うまでもない。例えば、記載の実施形態はＪａｖａ仮想マシンに関するものだが、本発明の原理は他のシステムや言語に対して容易に適用できる。従って、上記の説明は、付属の請求項の境界ならびに同等仕様の全範囲によって定義された本発明の適用範囲を限定するものと見做してはならない。

【図面の簡単な説明】

【図１】本発明の実施形態のソフトウェアを実行するために利用可能なコンピュータシステムの例を示した図である。

【図２】図１のコンピュータシステムのシステムブロック図を示した図である。

【図３】Ｊａｖａソースコードプログラムが実行される様態を示した図である。

【図４】Ｊａｖａランタイムシステムの実装のコンポーネントを示した図である。

【図５】Ｊａｖａスタック上に保存される関数用のフレームを示した図である。

【図６】ランタイム中に得られるアクティブオブジェクトに関する情報を保存するプロセスを示した図である。

【図７】ランタイム中に得られるアクティブオブジェクトに関する情報を提示するプロセスを示した図である。

【図８】スナップショットを取ったときにアクティブだったインスタンスを持ったすべてのクラスのハイパーテキストドキュメントを示した図である。

【図９】スナップショットを取ったときにアクティブだったインスタンスを持ったすべてのクラスのハイパーテキストドキュメントを示した図である。

【図１０】スナップショットを取ったときにアクティブだったインスタンスを持ったすべてのクラスのハイパーテキストドキュメントを示した図である。

【図１１】スナップショットを取ったときにアクティブだったインスタンスを持ったすべてのクラスのハイパーテキストドキュメントを示した図である。

【図１２】スナップショットを取ったときにアクティブだったインスタンスを持ったすべてのクラスのハイパーテキストドキュメントを示した図である。

【図１３】スナップショットを取ったときにアクティブだったインスタンスを持った或るクラスのハイパーテキストドキュメントを示した図である。

【図１４】サブクラスを除くクラスのインスタンスのハイパーテキストドキュメントを示した図である。

【図１５】サブクラスを含むクラスのインスタンスのハイパーテキストドキュメントを示した図である。

【図１６】アクティブオブジェクトに関する情報のハイパーテキストドキュメントを示した図である。

【図１７】弱いリファレンスを除くアクティブオブジェクトに対するルートセットリファレンスのハイパーテキストドキュメントを示した図である。

【図１８】弱いリファレンスを含むアクティブオブジェクトに対するルートセットリファレンスのハイパーテキストドキュメントを示した図である。

【図１９】アクティブオブジェクトから到達可能なすべてのオブジェクトのハイパーテキストドキュメントを示した図である。

【図２０】アクティブオブジェクトから到達可能なすべてのオブジェクトのハイパーテキストドキュメントを示した図である。

【図２１】ルートセットのすべてのメンバのハイパーテキストドキュメントを示した図である。

【図２２】ルートセットのすべてのメンバのハイパーテキストドキュメントを示した図である。

【図２３】ルートセットのすべてのメンバのハイパーテキストドキュメントを示した図である。

【図２４】ルートセットのすべてのメンバのハイパーテキストドキュメントを示した図である。

【図２５】ルートセットのすべてのメンバのハイパーテキストドキュメントを示した図である。

【図２６】２つの異なるランタイムにおけるアクティブオブジェクト間の差分（例えば、新規インスタンス）を確定するプロセスを示した図である。

【図２７】クラスの新規インスタンスのハイパーテキストドキュメントを示した図である。

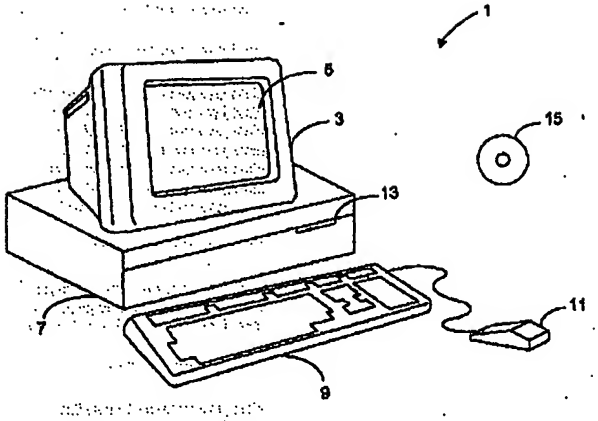
【図２８】サブクラスを含むクラスの新規インスタンスのハイパーテキストドキュメントを示した図である。

【図２９】アクティブオブジェクトに関する情報を保存するために利用可能なデータ構造を説明する。

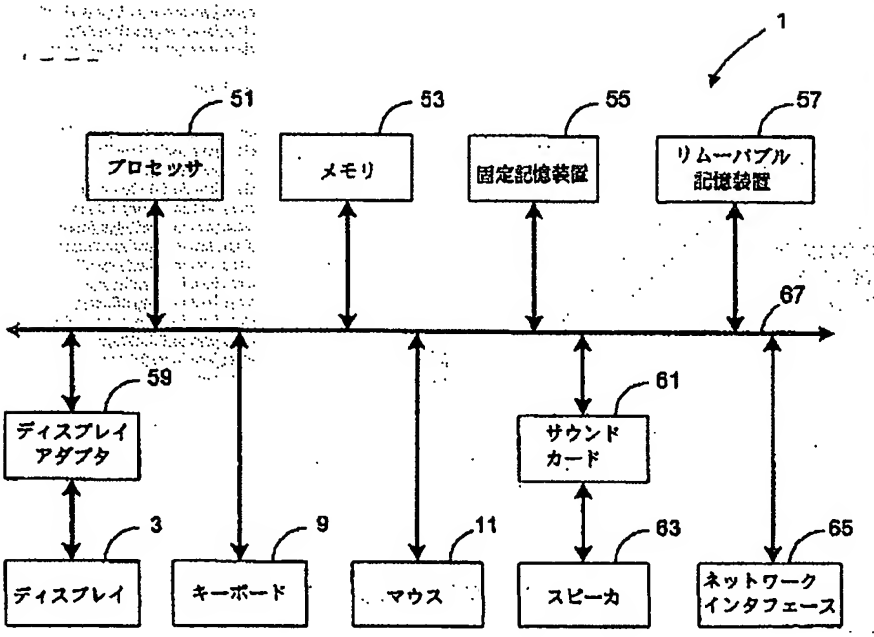
【図３０】アクティブオブジェクトに関する情報を保存するために利用可能なデータ構造を説明する。

【図３１】ＢＯＤサーバーの実施形態を示した図である。

【図1】



【図2】



【図8】

全クラス

Package <Default Package>

class char
class double
class float
class int

Package java.applet

class java.applet.Applet

Package java.awt

class java.awt.AWTEvent
class java.awt.BorderLayout
class java.awt.Canvas
class java.awt.Color
class java.awt.Component
class java.awt.Container
class java.awt.Cursor
class java.awt.Dimension
class java.awt.Event
class java.awt.EventDispatchThread
class java.awt.EventQueue
class java.awt.EventQueueItem
class java.awt.FlowLayout
class java.awt.FocusManager
class java.awt.Font
class java.awt.Frame
class java.awt.Graphics
class java.awt.GridBagConstraints
class java.awt.GridBagLayout
class java.awt.GridBagLayoutInfo
class java.awt.Image
class java.awt.Insets
class java.awt.LayoutManager
class java.awt.LayoutManager2
class java.awt.LightweightDispatcher
class java.awt.MenuComponent
class java.awt.Panel
class java.awt.Point
class java.awt.Polygon
class java.awt.Rectangle
class java.awt.ScrollPane
class java.awt.Shape
class java.awt.TextArea

【図12】

Package sun.io

class sun.io.CharToByte8859_1
class sun.io.CharToByteConverter
class sun.io.CharacterEncoding

Package <Arrays>
class [I

Other Queries

● Show all members of the root set

705

【図14】

jovial.slotCar.Car のインスタンス

class jovial.slotCar.Car

jovial.slotCar.Car@0xee704610 (24bytes)
jovial.slotCar.Car@0xee7043e0 (24bytes)

607

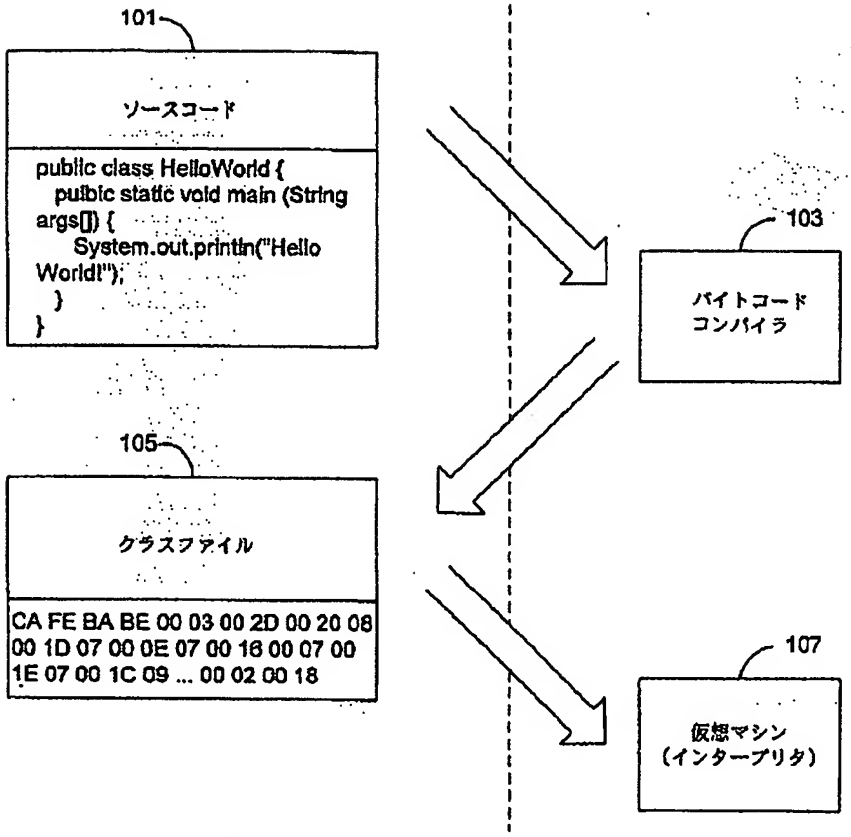
【図15】

.... jovial.slotCar.Car のインスタンス（サブクラスを含む）

class jovial.slotCar.Car

jovial.slotCar.Car@0xee704610 (24bytes)
jovial.slotCar.Car@0xee7043e0 (24bytes)

【図３】



【図９】

class java.awt.Window

Package java.awt.event

class java.awt.event.ComponentEvent
class java.awt.event.FocusEvent
class java.awt.event.InputEvent
class java.awt.event.KeyEvent
class java.awt.event.MouseEvent
class java.awt.event.PaintEvent
class java.awt.event.WindowEvent

Package java.awt.image

class java.awt.image.ImageConsumer
class java.awt.image.ImageObserver
class java.awt.image.ImageProducer

Package java.awt.peer

class java.awt.peer.CanvasPeer
class java.awt.peer.ComponentPeer
class java.awt.peer.ContainerPeer
class java.awt.peer.FontPeer
class java.awt.peer.FramePeer
class java.awt.peer.LightweightPeer
class java.awt.peer.PanelPeer
class java.awt.peer.WindowPeer

Package java.io

class java.io.BufferedInputStream
class java.io.BufferedWriter
class java.io.File
class java.io.FileDescriptor
class java.io.FileInputStream
class java.io.FileOutputStream
class java.io.FilterInputStream
class java.io.FilterOutputStream
class java.io.InputStream
class java.io.OutputStream
class java.io.OutputStreamWriter
class java.io.PrintStream
class java.io.Serializable
class java.io.Writer

【図１６】

0xee704610 のオブジェクト

instance of jovial.slotCar.Car (24 bytes)

Class:

class jovial.slotCar.Car

Instance data members:

driveColor [I,java/awt/Color] :java.awt.Color@0xee703108 (12 bytes)
crazyColor [I,java/awt/Color] :java.awt.Color@0xee703168 (12bytes)
gasPedal [I,jovial/slotCar/GasPedal] :jovial.slotCar.RandomGasPedal@0xee704608 (20 bytes)
poly [I,java/awt/Polygon] :java.awt.Polygon@0xee704650 (20 bytes)
pos [I,jovial/slotCar/track/TrackPosition] :jovial.slotCar.track.TrackPosition@0xee704620 (32 bytes)

References to this object:

array@0xee704408 (12 bytes) : Element 1 of array
array@0xee704910 (44 bytes) : Element 7 of array

Other Queries

Reference Chains from Rootset

635
- Exclude weak refs
- Include weak refs

675
Objects reachable from here

681

【図１７】

jovial.slotCar.Car に対するルートセットリファレンス
（弱いリファレンスを除く）

References to jovial.slotCar.Car@0xee704610 (24 bytes)

Java Local References

Java stack local (from java.lang.Thread):

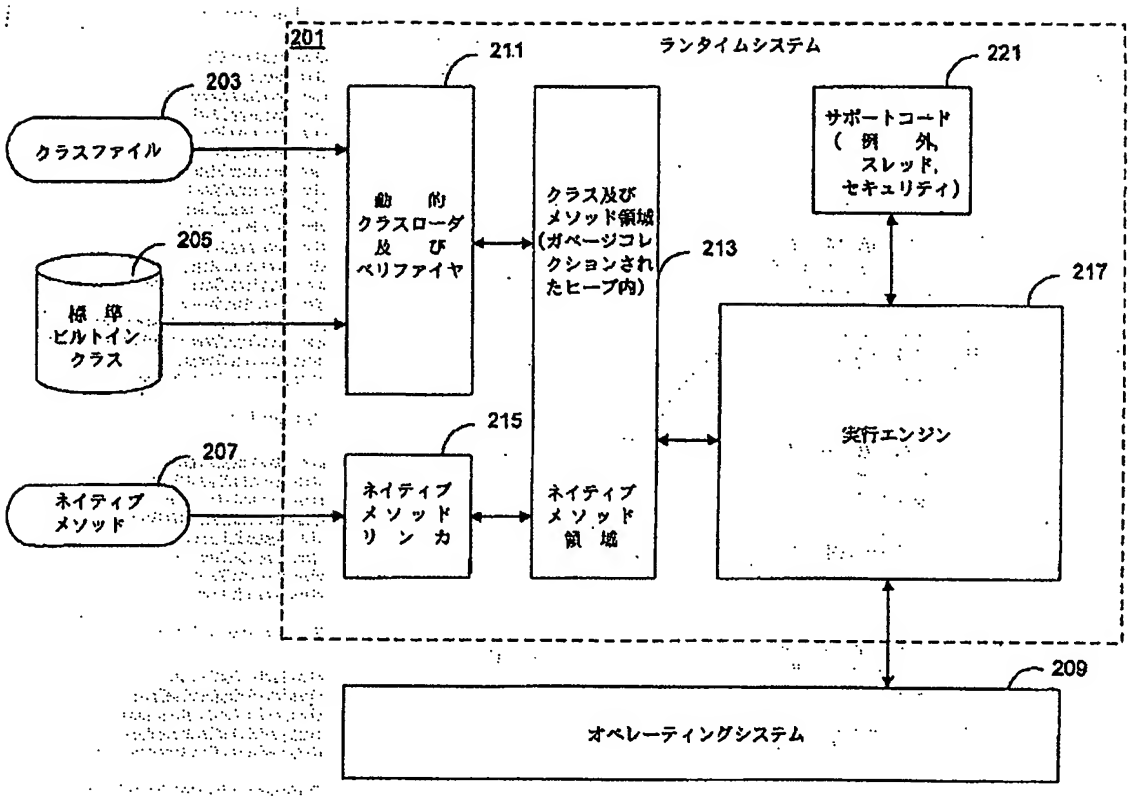
- jovial.slotCar.animator.Animator@0xee704908 (120 bytes) (field animator_)
- jovial.slotCar.RaceThread@0xee703f58 (80bytes) (field cars_)
- array@0xee704408 (12 bytes) (Element 1 of array:)
- jovial.slotCar.Car@0xee704610 (24 bytes)

Other queries

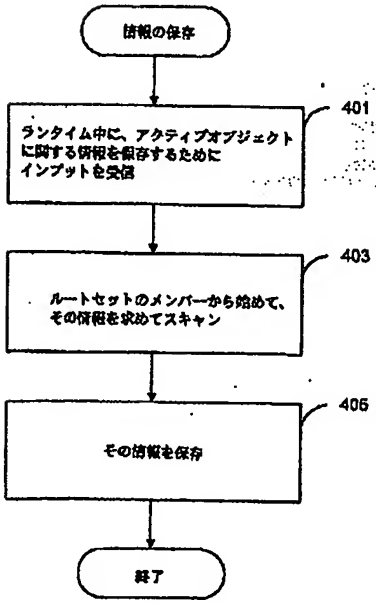
Include weak refs

691

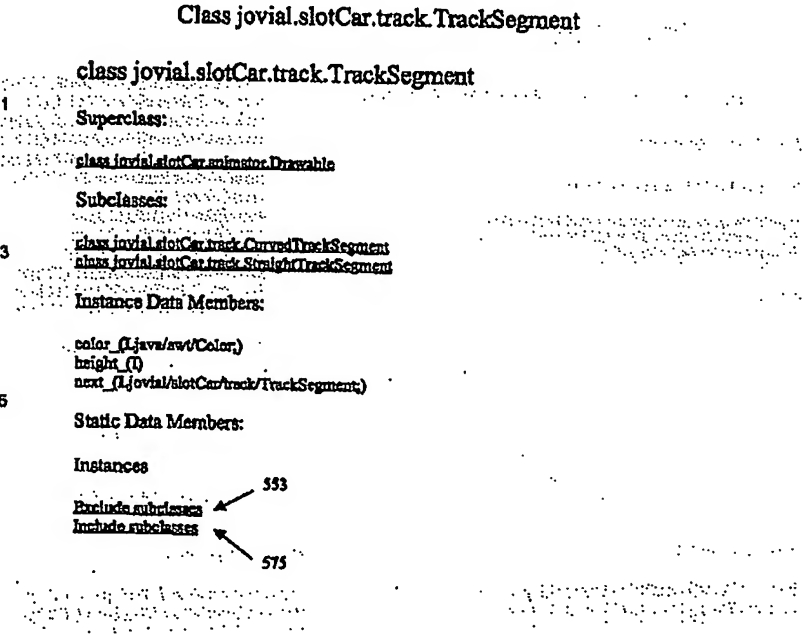
【図４】


201
ランタイムシステム
211
221
203
クラスファイル
動的クラスローダ及びベリファイヤ
クラス及びメソッド領域(ガベージコレクションされたヒープ内)
サポートコード(例外,スレッド,セキュリティ)
205
標準ビルトインクラス
213
217
実行エンジン
215
207
ネイティブメソッド
ネイティブメソッドリンカ
ネイティブメソッド領域
209
オペレーティングシステム

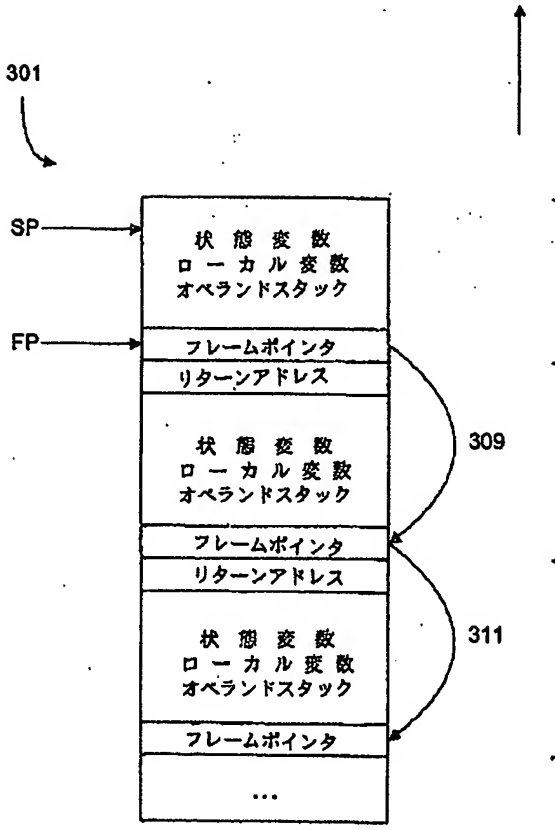
【図６】


情報の保存
ランタイム中に、アクティブオブジェクトに関する情報を保存するためにインプットを受信
401
ルートセットのメンバーから始めて、その情報を求めてスキャン
403
その情報を保存
405
終了

【図１３】


Class jovial.slotCar.track.TrackSegment
class jovial.slotCar.track.TrackSegment
Superclass:
class jovial.slotCar.animator.Drawable
Subclasses:
class jovial.slotCar.track.CurvedTrackSegment
class jovial.slotCar.track.StraightTrackSegment
Instance Data Members:
color_(Ljava/awt/Color;)
height_(I)
next_(Ljovial/slotCar/track/TrackSegment;)
Static Data Members:
Instances
553
Exclude subclasses
Include subclasses
575

【図５】



【図１０】

Package java.lang

class java.lang.Character
class java.lang.Class
class java.lang.Cloneable
class java.lang.Double
class java.lang.Float
class java.lang.Integer
class java.lang.Math
class java.lang.Number
class java.lang.Object
class java.lang.Runnable
class java.lang.Runtime
class java.lang.String
class java.lang.StringBuffer
class java.lang.System
class java.lang.System$DelegatingInputStream
class java.lang.System$DelegatingPrintStream
class java.lang.Thread
class java.lang.ThreadGroup

Package java.util

class java.util.Dictionary
class java.util.EventObject
class java.util.Hashtable
class java.util.HashtableEntry
class java.util.Locale
class java.util.Properties
class java.util.Random
class java.util.Vector

Package jovial.slotCar

class jovial.slotCar.Car
class jovial.slotCar.GasPedal
class jovial.slotCar.RaceApplet
class jovial.slotCar.RaceView
class jovial.slotCar.RandomGasPedal

Package jovial.slotCar.animator

class jovial.slotCar.animator.Animates
class jovial.slotCar.animator.Animator
class jovial.slotCar.animator.Drawable

【図１８】

jovial.slotCar.Car に対するルートセットリファレンス
(間いリファレンスを含む)

References to jovial.slotCar.Car@0xee704610 (24 bytes)

Java Local References

Java stack local (from java.lang.Thread):

→ jovial.slotCar.animator.Animator@0xee704658 (120 bytes) (field animatee_)
→ jovial.slotCar.RaceView@0xee703f58 (8 bytes) (field cars_)
→ array@0xee7044f8 (12 bytes) (Element 1 of array)
→ jovial.slotCar.Car@0xee704610 (24 bytes)

Other queries

Reachable weak refs

693

【図２０】

java.awt.Polygon@0xee7044f8 (20 bytes)
java.awt.Polygon@0xee704440 (20 bytes)
java.awt.Polygon@0xee704118 (20 bytes)
java.awt.Polygon@0xee704378 (20 bytes)
java.awt.Polygon@0xee704478 (20 bytes)
jovial.slotCar.RandomGasPedal@0xee704608 (20 bytes)
array@0xee704390 (12 bytes)
array@0xee7041a0 (12 bytes)
array@0xee7042d8 (12 bytes)
array@0xee7044c8 (12 bytes)
java.awt.Color@0xee703fa8 (12 bytes)
java.awt.Color@0xee703fc8 (12 bytes)
java.awt.Point@0xee7056a18 (12 bytes)

Total size: 1472 bytes.

【図２７】

java.awt.Point の新規インスタンス

class java.awt.Point ← 801

java.awt.Point@0xee706e58[new] (12 bytes)
java.awt.Point@0xee706e48[new] (12 bytes)
java.awt.Point@0xee706e08[new] (12 bytes)
803

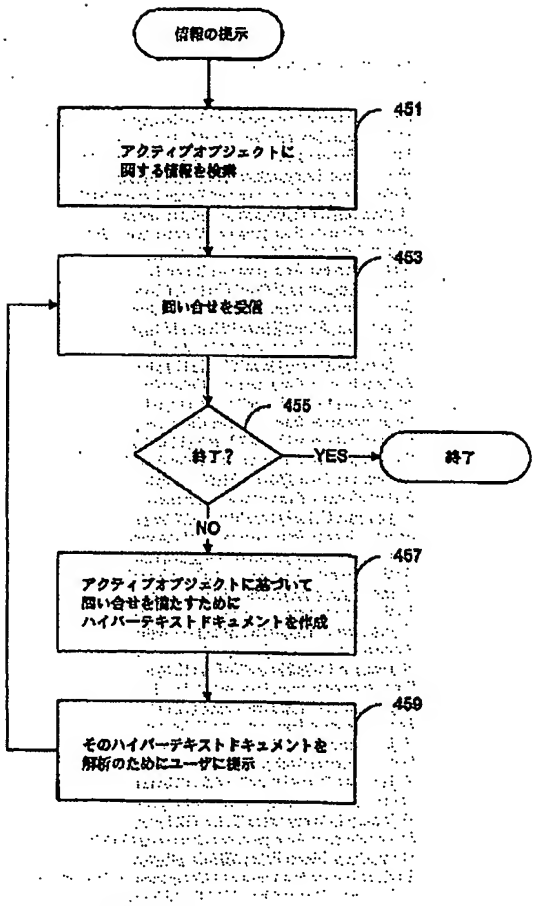
【図２８】

java.lang.Object の新規インスタンス (サブクラスを含む)

class java.lang.Object

java.awt.Point@0xee7056a18[new] (12 bytes)
java.awt.Point@0xee706e48[new] (12 bytes)
java.awt.Point@0xee706e08[new] (12 bytes)

【図7】



【図11】

Package jovial.slotCar.track

class jovial.slotCar.track.CurvedTrackSegment
class jovial.slotCar.track.FigureEightTrack
class jovial.slotCar.track.StraightTrackSegment
class jovial.slotCar.track.Track
class jovial.slotCar.track.TrackPosition
class jovial.slotCar.track.TrackSegment 503

Package sun.awt

class sun.awt.AWTFinalizable
class sun.awt.AWTFinalizer
class sun.awt.CharToByteSymbol
class sun.awt.DrawingSurface
class sun.awt.EmbeddedFrame
class sun.awt.FontDescriptor
class sun.awt.PlatformFont
class sun.awt.ScreenUpdater
class sun.awt.ScreenUpdaterEntry
class sun.awt.SunToolkit
class sun.awt.UpdateClient

Package sun.awt.image

class sun.awt.image.Image
class sun.awt.image.ImageFetchable
class sun.awt.image.ImageRepresentation
class sun.awt.image.ImageWatched
class sun.awt.image.InputStreamImageSource
class sun.awt.image.OffScreenImageSource

Package sun.awt.motif

class sun.awt.motif.CharToByteX11Dingbats
class sun.awt.motif.InputThread
class sun.awt.motif.MCanvasPeer
class sun.awt.motif.MComponentPeer
class sun.awt.motif.MEmbeddedFrame
class sun.awt.motif.MFontPeer
class sun.awt.motif.MFramePeer
class sun.awt.motif.MPanelPeer
class sun.awt.motif.MToolkit
class sun.awt.motif.X11Graphics
class sun.awt.motif.X11Image
class sun.awt.motif.X11OffScreenImage

【図25】

→ java.lang.Thread@0xee7052b0 (48 bytes)
Native code reference (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee7077c0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee70778 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.awt.Point@0xee706c08 (12 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee70778 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7052b0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee70778 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703a08 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703a08 (48 bytes)

Other Queries

● Show All Classes 715

【図30】

```
class ::= superclass_id : int
          name_len : int
          num_statics : int
          statics : static_field[]
          num_fields : int
          fields : field[]
          instance_size : int              // in bytes, including VM overhead

field ::= name_len : int
          name : char[]
          signature_len : int
          signature : char[]

static_field ::= field
                 value : int               // value is value of int field, or id of object

object_array ::= size : int                // in bytes, including VM overhead
                 len : int
                 object_id : int[]

char_array ::= size : int
               len : int
               data : char[]

Other_array ::= size : int
```

【図１９】

0xee704810 から到達可能なオブジェクト

jovial.slotCar.Car@0xee704610 (24 bytes)

jovial.slotCar.track.StraightTrackSegment@0xee704398 (88 bytes)
jovial.slotCar.track.StraightTrackSegment@0xee7041b8 (88 bytes)
jovial.slotCar.track.StraightTrackSegment@0xee704460 (88 bytes)
jovial.slotCar.track.StraightTrackSegment@0xee7043b8 (88 bytes)
jovial.slotCar.track.CurvedTrackSegment@0xee7043a0 (60 bytes)
jovial.slotCar.track.CurvedTrackSegment@0xee7041c8 (60 bytes)
jovial.slotCar.track.CurvedTrackSegment@0xee704418 (60 bytes)
jovial.slotCar.track.TrackPosition@0xee704620 (32 bytes)
<array>@0xee704138 (20 bytes)
<array>@0xee704130 (20 bytes)
<array>@0xee704460 (20 bytes)
<array>@0xee704358 (20 bytes)
<array>@0xee7045d0 (20 bytes)
<array>@0xee7043b0 (20 bytes)
<array>@0xee704448 (20 bytes)
<array>@0xee7045d8 (20 bytes)
<array>@0xee7042e8 (20 bytes)
<array>@0xee7043e8 (20 bytes)
<array>@0xee704420 (20 bytes)
<array>@0xee704168 (20 bytes)
<array>@0xee7044e0 (20 bytes)
<array>@0xee704268 (20 bytes)
<array>@0xee7042a8 (20 bytes)
<array>@0xee704170 (20 bytes)
<array>@0xee7042e0 (20 bytes)
<array>@0xee704360 (20 bytes)
<array>@0xee704490 (20 bytes)
<array>@0xee7042f0 (20 bytes)
<array>@0xee704320 (20 bytes)
<array>@0xee704328 (20 bytes)
<array>@0xee704498 (20 bytes)
<array>@0xee704510 (20 bytes)
<array>@0xee704428 (20 bytes)
<array>@0xee704270 (20 bytes)
java.awt.Polygon@0xee704288 (20 bytes)
java.awt.Polygon@0xee704308 (20 bytes)
java.awt.Polygon@0xee7042c0 (20 bytes)
java.awt.Polygon@0xee704340 (20 bytes)
java.awt.Polygon@0xee704260 (20 bytes)
java.awt.Polygon@0xee704188 (20 bytes)
java.awt.Polygon@0xee7044b0 (20 bytes)
java.awt.Polygon@0xee7045f0 (20 bytes)

【図２１】

ルートセットの全メンバー

Java Static References

Static reference from java.awt.BorderLayout.CENTER:
→ java.lang.String@0xee703e58 (16 bytes)
Static reference from java.awt.BorderLayout.EAST:
→ java.lang.String@0xee703a78 (16 bytes)
Static reference from java.awt.BorderLayout.NORTH:
→ java.lang.String@0xee703c98 (16 bytes)
Static reference from java.awt.BorderLayout.SOUTH:
→ java.lang.String@0xee703a88 (16 bytes)
Static reference from java.awt.BorderLayout.WEST:
→ java.lang.String@0xee703e68 (16 bytes)
Static reference from java.awt.Canvas.base:
→ java.lang.String@0xee7040b0 (16 bytes)
Static reference from java.awt.Color.black:
→ java.awt.Color@0xee703fb8 (12 bytes)
Static reference from java.awt.Color.blue:
→ java.awt.Color@0xee704240 (12 bytes)
Static reference from java.awt.Color.cyan:
→ java.awt.Color@0xee704248 (12 bytes)
Static reference from java.awt.Color.darkGray:
→ java.awt.Color@0xee703fd0 (12 bytes)
Static reference from java.awt.Color.gray:
→ java.awt.Color@0xee703fd8 (12 bytes)
Static reference from java.awt.Color.green:
→ java.awt.Color@0xee703fb0 (12 bytes)
Static reference from java.awt.Color.lightGray:
→ java.awt.Color@0xee703fe0 (12 bytes)
Static reference from java.awt.Color.magenta:
→ java.awt.Color@0xee704250 (12 bytes)
Static reference from java.awt.Color.orange:
→ java.awt.Color@0xee703fb0 (12 bytes)
Static reference from java.awt.Color.pink:
→ java.awt.Color@0xee703fb8 (12 bytes)
Static reference from java.awt.Color.red:
→ java.awt.Color@0xee703fc0 (12 bytes)
Static reference from java.awt.Color.white:
→ java.awt.Color@0xee703fc8 (12 bytes)
Static reference from java.awt.Color.yellow:
→ java.awt.Color@0xee703fa8 (12 bytes)
Static reference from java.awt.Component.LOCK
→ java.lang.Object@0xee703d20 (4bytes)
Static reference from java.awt.Component.actionListenerK:
→ java.lang.String@0xee703dd8 (16 bytes)

【図22】

Static reference from java.awt.Component.adjustmentListenerK:
→ java.lang.String@0xee703dc8 (16 bytes)
Static reference from java.awt.Component.componentListenerK:
•
•
•

Static reference from sun.io.CharToByteConverter.pkgString:
→ java.lang.String@0xee700538 (16 bytes)
Static reference from sun.io.CharacterEncoding.aliasTable:
→ java.util.Hashtable@0xee7005c8 (20 bytes)

Java Local References

Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.X11Graphics@0xee707a60 (36 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventDispatchThread@0xee7052c8 (56 bytes)
Java stack local (from java.lang.Thread):
→ jovial.slotCar.animator.Animator@0xee704058 (120 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.MToolkit@0xee705368 (8 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ java.lang.ThreadGroup@0xee703bc8 (44 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.event.FocusEvent@0xee707648 (24 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ jovial.slotCar.animator.Animator@0xee704058 (120 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.MToolkit@0xee705368 (8 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.X11Graphics@0xee707a48 (36 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.event.FocusEvent@0xee707648 (24 bytes)
Java stack local (from java.lang.Thread):

【図23】

→ sun.awt.motif.MCanvasPeer@0xee707580 (16 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707c08 (48 bytes)
Java stack local (from java.lang.Thread):
→ java.lang.Thread@0xee7052b0 (48 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.X11OffScreenImage@0xee707ad0 (52 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707c08 (48 bytes)
Java stack local (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707c08 (48 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707c08 (48 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)

Native Static References

Native code reference:
→ sun.awt.AWTFinalizer@0xee707c08 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee7000b0 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee703e20 (48 bytes)
Native code reference:
→ java.awt.EventDispatchThread@0xee7052c8 (56 bytes)
Native code reference:
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Native code reference:
→ java.lang.Thread@0xee703c48 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee700088 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference:
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee703cd8 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee7052b0 (48 bytes)

【図29】

BODファイルの構造
@(#)bod format.txt

このドキュメントは、bod（「バイナリーオブジェクトダンプ」）ファイルのファイルフォーマットを示す。このバージョンでは、報告されるデータ型は int、char 配列、及びレファレンスを含む一一他のタイプのフィールドは単に報告されない。これは非公式なBNFスタイルの構文で記述され、ときには文法中の符号はそれ自体（すなわち、「クラス」）によって与えられ、またときには記述ネーム(descriptive name)とコロンの後で与えられる（すなわち、static_field[]）。記号構文は、「このエレメントのゼロ以上」を意味する。「物(thing)」（クラスまたはオブジクト等）は、ダンプファイル内に２回以上現われるかもしれない。このルールの意図は主として、ファイルを作成するVM実装をより容易にするためである。物が３回現われる場合、多分 root_flags と thread_id を除いて、他点は同値を持つものと予想される。オブジェクトのルートのセットは、ファイル内のオブジェクトのすべての出現によって特定されるルートの集合(union)中のオブジェクトのすべての出現によって特定されるルートの集合である（すなわち、オブジェクトが２つの異なるスレッドの２つのローカル変数によって指示される場合、各ローカル可達ルートを示すエントリーが少なくとも１回現われる中で、そのオブジェクトはそのファイルの中に少なくとも２回現われる）。

bod file ::= magic_number version_number thing[]

magic_number ::= 0x0b0d000d

version_number ::= 4

thing ::= type:byte id:int root_flags:byte thread_id:int stuff

type is object=0, class=1, object array=2, char array=3, other array=4
root_flags is static=1, java stack local=2, native ref=4
thread_id is id field of the thread object responsible for this being a root, or 0 if not applicable

stuff ::= object | class | object_array | char_array

object ::= class_id : int
num_fields :int
data : int[]

data is the value of int fields, or the id of object/array fields

【図24】

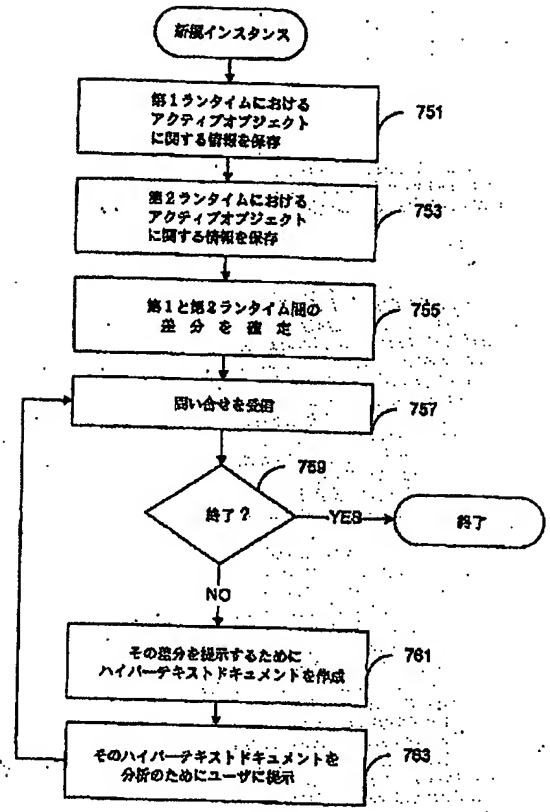
Native Local References

Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Native code reference(from java.lang.Thread):
→ sun.awt.motif.MToolkit@0xee703568 (8bytes)
Native code reference(from java.lang.Thread):
→ java.lang.Thread@0xee703c48 (48 bytes)
Native code reference (from java.lang.Thread):
→ array@0xee703f88 (4 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Native code reference (from java.lang.Thread):
→ array@0xee703f88 (4 bytes)
Native code reference (from sun.awt.motif.InputThread):

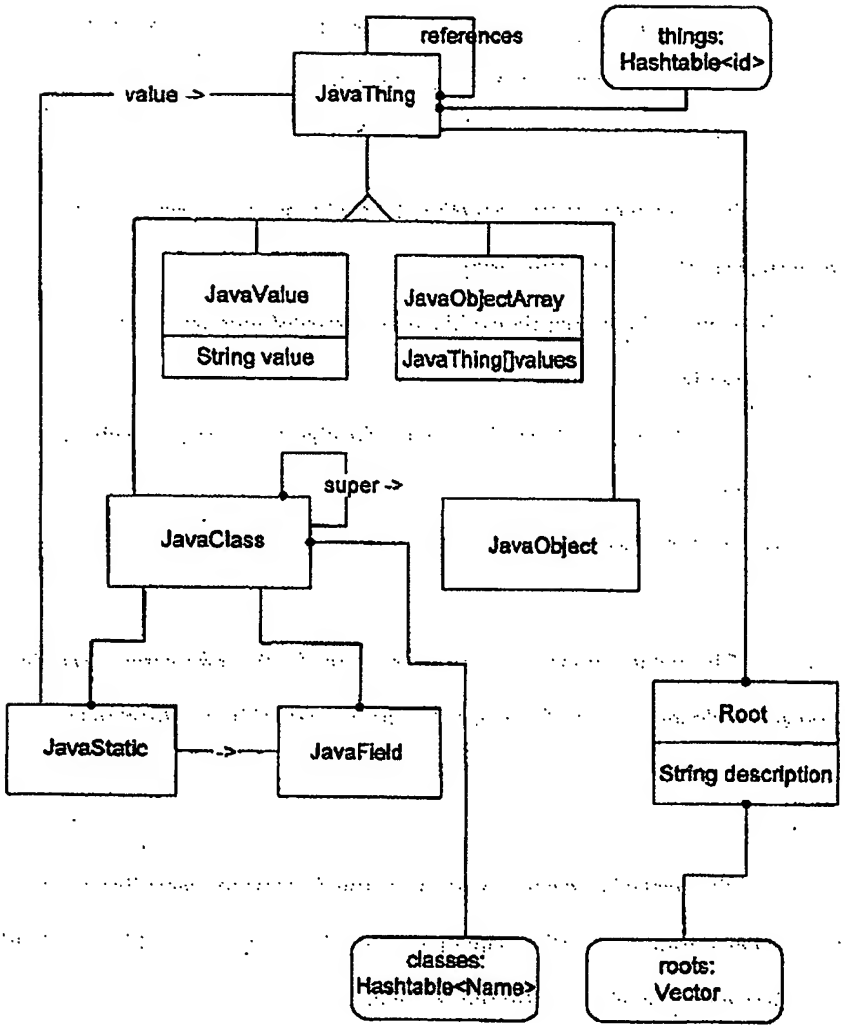
•
•
•

→ sun.awt.AWTFinalizer@0xee707c08 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventDispatchThread@0xee7052e8 (56 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.Event@0xee707078 (56 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.Event@0xee7075c8 (56 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7052b0 (48 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.lang.Thread):

【図26】



【図３１】



フロントページの続き

(72)発明者 ジェフリー ディー. ナイズワンガー
アメリカ合衆国, カリフォルニア州,
サン ノゼ, ノース 16ティーエイチ
ストリート 532

【外国語明細書】

1 Title of Invention

OBJECT HEAP ANALYSIS TECHNIQUES FOR DISCOVERING MEMORY LEAKS AND OTHER RUN-TIME INFORMATION

2 Claims

1. In a computer system, a method for analyzing the execution of object-oriented programs, the method comprising:

receiving input during run-time of the object-oriented program to store information regarding active objects;

scanning for the active objects beginning with objects that are a member of a root set of objects; and

storing the information regarding active objects.

2. The method of claim 1, wherein scanning for the information regarding active objects includes scanning for objects that are referenced by an object in the root set of objects.

3. The method of claim 1, wherein storing information regarding active objects includes storing information regarding objects in the root set of objects and objects that are referenced by an object in the root set of objects.

4. The method of claim 1 or 2, wherein the information regarding active objects includes an indication of the objects that are members of the root set of objects.

5. The method of any of the preceding claims, wherein each object in the root set of objects is referenced by a static data member, a local reference from a stack or a reference from native code.

6. The method of any of the preceding claims, wherein the active objects include objects that define the structure of other objects.

7. The method of any of the preceding claims, wherein the information regarding objects that are objects are stored in a file.

8. The method of any of the preceding claims, wherein the active objects include instances of Java classes.

9. A computer program product for analyzing the execution of object-oriented programs, comprising:

computer code that receives input during run-time of the object-oriented program to store information regarding active objects;

computer code that scans for the active objects beginning with objects that are a member of a root set of objects;

computer code that stores the information regarding active objects; and

a computer readable medium that stores the computer codes.

10. The computer program product of claim 9, wherein the computer code that scans for the information regarding active objects includes computer code that scans for objects that are referenced by an object in the root set of objects.

11. The computer program product of claim 9 or 10, wherein the computer code that stores the information regarding active objects includes computer code that

stores information regarding objects in the root set of objects and objects that are referenced by an object in the root set of objects.

12. The computer program product of any of claims 9-11, wherein the information regarding active objects includes an indication of the objects that are a member of the root set of objects.

13. The computer program product of any of claims 9-12, wherein each object in the root set of objects is referenced by a static data member, a local reference from a stack or a reference from native code.

14. The computer program product of any of claims 9-13, wherein the active objects include objects that define the structure of other objects.

15. The computer program product of any of claims 9-14, wherein the information regarding objects that are objects are stored in a file.

16. The computer program product of any of claims 9-15, wherein the active objects include instances of Java classes.

17. The computer program product of any of claims 9-16, wherein the computer readable medium is selected from the group consisting of CD-ROM, floppy disk, tape, flash memory, system memory, hard drive, and data signal embodied in a carrier wave.

18. In a computer system, a method for analyzing the execution of object-oriented programs, the method comprising:

retrieving information regarding active objects at a first point in time of the execution of an object-oriented program;

generating a hypertext document to present the information regarding active objects; and

presenting the hypertext document to a user for analysis.

19. The method of claim 18, further comprising retrieving information regarding active objects at a second point in time of the execution of an object-oriented program.

20. The method of claim 19, further comprising determining differences between active objects at the first and second points in time.

21. The method of claim 20, wherein the differences include active objects that are present at the second point in time but not at the first point in time.

22. The method of any of claims 18-21, wherein the active objects includes instances of Java classes and the hypertext document is written in the HyperText Markup Language (HTML).

23. A computer program product for analyzing the execution of object-oriented programs, comprising:

computer code that retrieves information regarding active objects at a first point in time of the execution of an object-oriented program;

computer code that generates a hypertext document to present the information regarding active objects;

computer code that presents the hypertext document to a user for analysis; and

a computer readable medium that stores the computer codes.

24. The computer program product of claim 23, further comprising:

computer code that retrieves information regarding active objects at a second point in time of the execution of an object-oriented program; and

computer code that determines differences between active objects at the first and second points in time, wherein the differences include active objects that are present at the second point in time but not at the first point in time.

25. In a computer system, a method for analyzing the execution of object-oriented programs, the method comprising:

during run-time of an object-oriented program, storing information regarding active objects at first and second points in time of the execution of an object-oriented program;

retrieving the information regarding active objects at the first and second points in time of the execution of an object-oriented program;

determining differences between active objects at the first and second points in time;

generating a hypertext document to present the differences between active objects at the first and second points in time; and

presenting the hypertext document to present the differences between active objects at the first and second points in time to a user for analysis.

26. A computer program product for analyzing the execution of object-oriented programs, comprising:

computer code that, during run-time of an object-oriented program, stores information regarding active objects at first and second points in time of the execution of an object-oriented program;

computer code that retrieves the information regarding active objects at the first and second points in time of the execution of an object-oriented program;

computer code that determines differences between active objects at the first and second points in time;

computer code that generates a hypertext document to present the differences between active objects at the first and second points in time; and

computer code that presents the hypertext document to present the differences between active objects at the first and second points in time to a user for analysis; and

a computer readable medium that stores the computer codes.

3 Detailed Description of Invention

BACKGROUND OF THE INVENTION

The present invention relates to techniques for analyzing (e.g., optimization, locating errors or "debugging") software applications. More specifically, the invention relates to techniques for storing information about objects existing during run-time of an application on a Java™ virtual machine for later analysis.

The Java™ programming language is an object-oriented high level programming language developed by Sun Microsystems and designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up to supercomputers. Computer programs written in Java (and other languages) may be compiled into virtual machine instructions for execution by a Java virtual machine. In general the Java virtual machine is an interpreter that decodes and executes the virtual machine instructions.

The virtual machine instructions for the Java virtual machine are bytecodes, meaning they include one or more bytes. The bytecodes are stored in a particular file format called a "class file" that includes bytecodes for methods of a class. In addition to the bytecodes for methods of a class, the class file includes a symbol table as well as other ancillary information.

A computer program embodied as Java bytecodes in one or more class files is platform independent. The computer program may be executed, unmodified, on any computer that is able to run an implementation of the Java virtual machine. The Java virtual machine is a software emulator of a "generic" computer that is a major factor in allowing computer programs for the Java virtual machine to be platform independent.

The Java virtual machine is commonly implemented as a software interpreter. Conventional interpreters decode and execute the virtual machine instructions of an interpreted program one instruction at a time during execution, which is in contrast to compilers that decode source code into native machine instructions prior to execution so that decoding is not performed during execution. Typically, the Java virtual machine will be written in a programming language other than the Java programming language (e.g., the C++ programming language). Therefore, execution of a Java program may involve execution of functions written in multiple programming languages. Additionally, the bytecodes themselves may call functions (e.g., system functions for input/output) that are not written in the Java programming language. It is therefore common for an executing Java program to entail the execution of functions that were written in multiple programming languages.

Although it is a goal of object-oriented programs to allow for the reuse of tested source code and therefore a reduction in the number of run-time errors, Java programs may still benefit from analysis techniques that provide a window into the operation of the program during run-time. For example, the analysis may be utilized to optimize the program or locate bugs in the code. Accordingly, it would be desirable to provide innovative techniques of analyzing applications executing on a Java virtual machine. Additionally, it would be beneficial to provide a snapshot of objects that exist during run-time of an application so that, for example, memory leaks may be detected.

SUMMARY OF THE INVENTION

In general, embodiments of the present invention provide innovative techniques for analyzing object-oriented computer programs. A snapshot of the objects that are active ("active objects") at a specific point in time during execution may be stored. An analysis tool may be utilized to generate hypertext documents that allow a user to analyze the active objects. As an example, a user may be able to identify memory leaks by browsing through HyperText Markup Language (HTML) documents and determining that there are too many instances of a class than is expected. The user may then follow the references (or pointers) to the unnecessary instances in order to determine what is causing the memory leak. Additionally, a user may compare two different snapshots of active objects at two different run-times so that, for example, new instances of a class may be easily identified. Several embodiments of the invention are described below.

In one embodiment, a computer-implemented method for analyzing the execution of an object-oriented program includes receiving input during run-time to store information regarding active objects. In order to locate the active objects, the system may begin scanning active objects beginning with objects that are a member of the root set of objects. Then, objects that referenced by the root set of objects may be identified, and objects that are referenced by these objects, and so on until all the active objects are identified. Information regarding the active objects may be stored (e.g., in a file). In preferred embodiments, the active objects are instances of Java classes.

In another embodiment, a computer-implemented method for analyzing the execution of an object-oriented program includes retrieving information regarding active objects at a point in time in the execution. The information may be recursively scanned to identify objects that are members of the root set and maintain the hierarchy of the active

objects. A hypertext document may be generated to present the information regarding the active objects to a user for analysis.

In another embodiment, a computer-implemented method for analyzing the execution of an object-oriented program includes storing information regarding active objects at two different run-times. The differences between the first and second run-times, e.g., new instances, may then be determined. A hypertext document may be generated to present the differences to a user for analysis.

Other features and advantages of the invention will become readily apparent upon review of the following detailed description in association with the accompanying drawings.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Definitions

Class – An object-oriented data type that defines objects that share the same characteristics, typically including both data and functions that operate on the data.

Object (or instance) – An instantiated member of a class.

Root set of objects – Objects that may be referenced directly without chaining through other objects.

Active object – An object that has been instantiated and is a member of the root set or may be referenced by members of the root set either directly or indirectly.

Native methods (or code) – Functions that are written in a programming language other than the Java programming language.

Overview

In the description that follows, the present invention will be described in reference to preferred embodiments that analyze the execution of a Java program (e.g., bytecodes) during execution. In particular, examples will be described in which the Java virtual machine is written in the C++ programming language. However, the invention is not limited to any particular language, computer architecture, or specific implementation. Therefore, the description of the embodiments that follow is for purposes of illustration and not limitation.

FIG. 1 illustrates an example of a computer system that may be used to execute the software of an embodiment of the invention. FIG. 1 shows a computer system 1 that includes a display 3, screen 5, cabinet 7, keyboard 9, and mouse 11. Mouse 11 may have one or more buttons for interacting with a graphical user interface. Cabinet 7 houses a

CD-ROM drive 13, system memory and a hard drive (see FIG. 2) which may be utilized to store and retrieve software programs incorporating computer code that implements the invention, data for use with the invention, and the like. Although the CD-ROM 15 is shown as an exemplary computer readable storage medium, other computer readable storage media including floppy disk, tape, flash memory, system memory, and hard drive may be utilized. Additionally, a data signal embodied in a carrier wave (e.g., in a network including the Internet) may be the computer readable storage medium.

FIG. 2 shows a system block diagram of computer system 1 used to execute the software of an embodiment of the invention. As in FIG. 1, computer system 1 includes monitor 3 and keyboard 9, and mouse 11. Computer system 1 further includes subsystems such as a central processor 51, system memory 53, fixed storage 55 (e.g., hard drive), removable storage 57 (e.g., CD-ROM drive), display adapter 59, sound card 61, speakers 63, and network interface 65. Other computer systems suitable for use with the invention may include additional or fewer subsystems. For example, another computer system could include more than one processor 51 (i.e., a multi-processor system), or a cache memory.

The system bus architecture of computer system 1 is represented by arrows 67. However, these arrows are illustrative of any interconnection scheme serving to link the subsystems. For example, a local bus could be utilized to connect the central processor to the system memory and display adapter. Computer system 1 shown in FIG. 2 is but an example of a computer system suitable for use with the invention. Other computer architectures having different configurations of subsystems may also be utilized.

Typically, computer programs written in the Java programming language are compiled into bytecodes or Java virtual machine instructions that are then executed by a Java virtual machine. The bytecodes are stored in class files that are input into the Java

virtual machine for interpretation. FIG. 3 shows a progression of a simple piece of Java source code through execution by an interpreter, the Java virtual machine.

Java source code 101 includes the classic Hello World program written in Java. The source code is then input into a bytecode compiler 103 that compiles the source code into bytecodes. The bytecodes are virtual machine instructions as they will be executed by a software emulated computer. Typically, virtual machine instructions are generic (i.e., not designed for any specific microprocessor or computer architecture) but this is not required. The bytecode compiler outputs a Java class file 105 that includes the bytecodes for the Java program.

The Java class file is input into a Java virtual machine 107. The Java virtual machine is an interpreter that decodes and executes the bytecodes in the Java class file. The Java virtual machine is an interpreter, but is commonly referred to as a virtual machine as it emulates a microprocessor or computer architecture in software (e.g., the microprocessor or computer architecture that may not exist in hardware).

FIG. 4 shows the components of an implementation of a Java runtime system. Implementations of the Java virtual machine are known as Java runtime systems. A Java runtime system 201 may receive input of Java class files 203, standard built-in Java classes 205 and native methods 207 in order to execute a Java program. The standard built-in Java classes may be classes for objects such as threads, strings and the like. The native methods may be written in programming languages other than the Java programming language. The native methods are typically stored in dynamic link libraries (DLLs) or shared libraries.

The Java runtime system may also interface with an operating system 209. For example, input/output functions may be handled by the operating system, including providing the Java runtime system with interfaces to Java class files 203, standard built-in Java classes 205 and native methods 207.

A dynamic class loader and verifier 211 loads Java class files 203 and standard built-in Java classes 205 via operating system 209 into a memory 213. Additionally, the dynamic class loader and verifier may verify the correctness of the bytecodes in the Java class files, reporting any errors that are detected.

A native method linker 215 links in native methods 207 via operating system 209 into the Java runtime system and stores the native methods in memory 213. As shown, memory 213 may include a class and method area for the Java classes and a native method area for native methods. The class and method area in memory 213 may be stored in a garbage collected heap. As new objects are created, they are stored in the garbage collected heap. The Java runtime system, not the application, is responsible for reclaiming memory in the garbage collected heap when space is no longer being utilized.

At the heart of the Java runtime system shown in FIG. 4 is an execution engine 217. The execution engine carries out the instructions stored in memory 213 and may be implemented in software, hardware or a combination of the two. The execution engine supports object-oriented applications and conceptually, there are multiple execution engines running concurrently, one for each Java thread. Execution engine 217 may also utilize support code 221. The support code may provide functionality relating to exceptions, threads, security, and the like.

As a Java program executes, functions are sequentially called within each thread. For each thread there is an execution stack which stores frames for each of the functions that have not completed execution. A frame stores information for execution of the function, such information may include state variables, local variables and an operand stack. As a function is called, a frame for the function is pushed on the execution stack. When the function terminates, the function's frame is popped off the execution stack. Accordingly, only the function corresponding to the frame on the top of the execution stack is active, the functions that correspond to frames below the top of the execution

stack have had their execution suspended until the function they called returns (*i.e.*, terminates).

FIG. 5 illustrates frames for Java functions that are stored on an execution stack. An execution stack 301 is shown having a frame 303 on the top of the execution stack and frames 305 and 307 stored below frame 303, respectively. A stack pointer SP points to the top of the execution stack while a frame pointer FP points to a frame pointer in the frame on the top of execution stack 301.

Each frame is shown to include state variables, local variables and an operand stack for the function corresponding to the frame. Additionally, the last item stored in the frame is a frame pointer which points to the frame pointer in the frame below the current frame on the execution stack as shown by arrows 309 and 311. Although the execution stack that stores frames for native methods may look different, the basic principles are typically similar to the execution stack shown.

Analyzing Run-time Execution

In order to analyze the execution of a Java program, the user executes the program with a Java virtual machine. The virtual machine is responsible for interpreting the Java program, and compilation may be performed for increased efficiency. FIG. 6 shows a process of storing information regarding active objects obtained during run-time of a Java program.

At a step 401, the system receives input during run-time to store information regarding active objects. For example, the user may depress certain keystrokes (e.g., <control> \) to indicate to the system that information regarding active objects should be stored.

The system then scans for the requested information beginning with members of the root set at a step 403. Objects that are members of the root set may be directly

referenced. For example, in preferred embodiments, objects that are members of the root set may be referenced by static data members of a Java class, referenced locally by the Java or native execution stack, or referenced globally by native code (global Java Native Interface reference). The system may start with objects that are members of the root set and recursively follow references to any other objects referenced by members of the root set. Thus, all objects that are members of the root set or may be referenced directly or indirectly from the members of the root set will be identified (i.e., transitive closure of reachable objects from the root set). These objects are the active objects at this point in time. In preferred embodiments, a mark and sweep algorithm is utilized similar to a garbage collection algorithm.

Once the active objects are identified, the system stores information about the objects at a step 405. The information may include an id of the object, flags indicating if the object is a member of the root set and why (e.g., referenced by static data members of a Java class), a thread id if applicable, class id, data of the object, and the like. Information that is stored in preferred embodiments is shown in FIGS. 18A-18B. In a Java program, objects of the class Class store the structure of other objects (i.e., the structure of all objects of a specific class) so the information regarding the active objects may include objects that define the structure of other objects.

The information regarding active objects may be stored in a file or any other computer storage (e.g., memory). The information may be stored when an active object is identified or subsequently all at one time. Therefore, the steps in the flowcharts are shown for illustration purposes and one of skill in the art would readily recognize that steps may be combined, deleted or inserted in other embodiments. Although the information may be analyzed during run-time, it is typically analyzed after the program finishes execution. FIG. 7 shows a process of presenting information regarding active objects obtained during run-time. At a step 451, the system retrieves stored information

regarding active objects. The information may include not only information about the active objects themselves, but also their relationships to each other. The relationships between active objects may resemble a directed graph structure that begins with members of the root set and typically proceeds to other objects through various levels of indirection. Thus, active objects that are not members of the root set may be identified by following references from a member of the root set in a manner similar to step 403 of FIG. 6.

Once the information regarding the active objects is retrieved, the system receives a user query at a step 453. In a preferred embodiment, the query is specified as a Uniform Resource Locator (URL) into a HyperText Transport Protocol (HTTP) server. The following will describe queries that may be available.

An "All Classes Query" shows all of the classes that were present on the heap at run-time. The classes may be sorted by their fully-qualified class name and organized by package. An example of the results of this query is shown in FIGS. 8A-8E.

A "Class Query" shows information about a desired class. The information may include the superclass, any subclasses, instance data members, and static data members. An example of the results of this query are shown in FIG. 9.

An "Instances Query" shows all the instances of a specified class. An example of the results of this query are shown in FIG. 10.

An "Object Query" shows information about an object that was on the heap at run-time. Most notably, one may navigate to objects that refer to this object, which may be utilized to track down errors. An example of the results of this query are shown in FIG. 12.

A "Roots Query" provides the reference chains from to root set to a specific object. A chain will be provided from each member of the root set from which the object of interest is reachable. In preferred embodiments, the chains are calculated by a depth-first search in order to reduce the length of the chains. Other search techniques may also

be utilized. The "Roots Query" is very valuable query for tracking down memory leaks as it may be utilized to determine why an object is still active. An example of the results of this query are shown in FIG. 13.

A "Reachable Objects Query" shows the transitive closure of all objects that are reachable from a specific object. This query may be useful for determining the total run-time footprint of an object in memory. An example of the results of this query are shown in FIG. (after FIG. 12).

An "All Roots Query" shows all the members of the root set. An example of the results of this query are shown in FIG. ???.

Unless the query indicates the user is finished at a step 455; the system generates a hypertext document or documents to present the information that fulfills the query at a step 455. A hypertext document includes links to other portions of the document or other documents altogether. In preferred embodiments, the hypertext document(s) is/are written in the HyperText Markup Language (HTML).

At a step 459, the system presents the hypertext document to a user for analysis. This may be done by creating an HTTP server for presenting an HTML document viewable with a Web browser.

The above has described embodiments of the invention but it may be helpful for the reader to see examples of the data that may be presented to a user for analysis. Once a snapshot of information regarding active objects is taken and put in an HTML document. A user may utilize a Web browser to analyze the information. A good starting point may be a query to show all the classes that had instances that were active.

FIGS. 8A-8E show a hypertext document of all classes that had instances that were active when the snapshot was taken. As shown, the class are organized by package. The words that are underlined represent hypertext links to other HTML documents. As an

example, a link 503 in FIG. 8D may be selected to see more details about the class jovial.slotCar.track.TrackSegment.

FIG. 9 shows a hypertext document of a class that had instances that were active when the snapshot was taken, which is in this case the class jovial.slotCar.track.TrackSegment. Thus, if a user "clicked" on link 503, the user would see the hypertext document shown in FIG. 9. As shown, information such as superclass, subclasses, instance data members, and static data members may be present with hypertext links to more detailed information.

At the bottom of the hypertext document in FIG. 9, a user may request to see the active instances of the class by a link 553 that excludes subclasses or a link 575 that includes subclasses. In other words, link 553 will show all instances of the class jovial.slotCar.track.TrackSegment but will not include instances of subclasses of this class. Link 575 will show all instances of the class jovial.slotCar.track.TrackSegment including instances of subclasses of this class (e.g., classes jovial.slotCar.track.CurvedTrackSegment and jovial.slotCar.track.StraightTrackSegment).

FIG. 10 shows a hypertext document of instances of a class excluding subclasses. Although the hypertext document would not be the one presented when link 553 is selected, it nevertheless represents an example of how the hypertext document may appear. As shown, there are two instances of class jovial.slotCar.Car that were located at the specified memory locations. If a user would like to analyze more information about either of these instances, the links may be selected. A link 607 will be described in more detail in reference to FIG. 12.

FIG. 11 shows a hypertext document of instances of a class including subclasses. This figure shows what may be presented if a user clicked on link 575 of FIG. 9. Since instances of subclasses are shown, some of the instances may include more data or

functions than jovial.slotCar.track.TrackSegment. However, all these instances will inherit from this parent class.

FIG. 12 shows a hypertext document of information regarding an active object. The active object is the one that may be selected by link 607 in FIG. 10. As shown, the class, instance data members, references to the object, and reference chains from the root set may be presented. As an example, a link 635 would show reference chains from the root set excluding weak references and a link 675 would show reference chains from the root set including weak references. A weak reference is a reference that the garbage collector may remove if more memory or heap space is needed. A link 681 would show all objects that are reachable (or referenced) by this object.

FIG. 13 shows a hypertext document of root set references to an active object that excludes weak references whereas FIG. 14 shows a hypertext document of root set references to an active object that includes weak references. Although no weak references are shown, a link 691 in FIG. 13 would allow a user to switch to FIG. 14 while a link 693 in FIG. 14 would allow the user to switch back.

FIGS. 15A and 15B show a hypertext document of all the objects that are reachable from an active object. This may be displayed when a user activates link 681 of FIG. 12. The objects may be identified by searching a directed graph of active objects.

Referring back to FIG. 8E, a link 705 allows a user to request to see all the members of the root set. FIGS. 16A-16E shows a hypertext document of all members of the root set that may be presented if link 705 is activated. As shown, the members of the root set may be organized according to how the members are referenced. For example, by Java static references, Java local references, native static references, and native local references. A link 715 in FIG. 16E allows a user to request all the classes that had active instances, such as FIGS. 8A-8E.

The hypertext document examples illustrate that a user may easily analyze the active objects in preferred embodiments. However, the invention is not limited to presentation by hypertext documents and one of skill in the art would realize other methods may be utilized.

In order to further understand the invention, it may be helpful to describe a scenario where a user would perform analysis of a Java program. Memory leaks are a pervasive problem in almost any programming language. A memory leak generally refers to memory that has been allocated but is no longer being used by a computer program. In worst case scenarios, memory leaks result in memory allocation errors because there is not enough available memory. A memory leak may be due to programmer error when there is a reference from the root set to an object in the heap that is no longer needed.

Assume a user suspects that there is a memory leakage problem. The user may count the number of instances of a class that should be active at a certain run-time. The user may then generate a snapshot of active objects according to an embodiment of the invention. The user may then analyze the stored information to determine how many instances were active. If there are more than expected, the user may follow the references from the errant active objects to determine what object is maintaining an erroneous reference.

Additionally, embodiments of the invention allow a user to compare snapshots of active objects. FIG. 17 shows a process of determining differences (e.g., new instances) between active objects at two different run-times.

At a step 751, the system stores information regarding active objects at a first run-time. Subsequently at a step 753, the system stores information regarding active objects at a second run-time. The system may be instructed to store the information by user input as described above.

A user may instruct the system to determine differences between the first and second run-times at a step 755. For example, the user may specify the files that store the snapshots at the two run-times. The system may then compare the snapshots and determine the differences between the snapshots, such as what new instances are present at the second run-time.

Once the differences between the first and second run-times is determined, the system receives a user query at a step 757. In addition to the queries described above, a "New Instances Query" may be available that shows only new instances in the second run-time. An instance may be considered "new" if it is in the first snapshot but there is no object with the same id in the second snapshot. An object's id may be a 32-bit integer (or handle) that is assigned by the virtual machine and uniquely identifies the object. Although handles may be re-used, snapshots that are taken within relatively short time intervals generally produce excellent results. An example of the results of this query are shown in FIG. 18.

Unless the query indicates the user is finished at a step 759, the system generates a hypertext document or documents to present the information that fulfills the query (e.g., the differences between the two snapshots) at a step 761. The system may present the hypertext document to a user for analysis at a step 763, typically by creating an HTTP server for presenting an HTML document viewable with a Web browser.

A hypertext document of new instances of a class is shown in FIG. 18. As shown, a link 801 allow a user to see more details about the class and links 803 allow a user to see more details about the new instances. In the hypertext document of FIG. 18, subclasses are not shown. FIG. 19 shows a hypertext document of new instances of a class that includes subclasses, of which there are no additional instances in this example.

During debugging, it may be beneficial to analyze how a computer program changes over time. With embodiments of the invention, a user is not only able to take

multiple snapshots of active objects during run-time, but is also able to have the system compare snapshots and present differences for further analysis.

To this point, the specific details about how the information regarding active objects may be stored. FIGS. 20A-20B describe a data structure that may be utilized to store information regarding active objects in preferred embodiments. A binary object dump (BOD) file may be stored according to the structure specified in these figures. Although this structure may be utilized in preferred embodiments, other structures may be utilized without departing from the spirit of the invention.

FIG. 21 shows a model part of a BOD server. When the server is started, it reads a BOD file and builds the structure shown in memory. The top level of this structure is an instance of Snapshot (the objects shown are data members of Snapshot). When a user requests information, the server treats the snapshot as a read-only representation of the contents of the heap.

Conclusion

While the above is a complete description of preferred embodiments of the invention, there is alternatives, modifications, and equivalents may be used. It should be evident that the invention is equally applicable by making appropriate modifications to the embodiments described above. For example, the embodiments described have been in reference to a Java virtual machine, but the principles of the present invention may be readily applied to other systems and languages. Therefore, the above description should not be taken as limiting the scope of the invention that is defined by the metes and bounds of the appended claims along with their full scope of equivalents.

4 Brief Description of Drawings

FIG. 1 illustrates an example of a computer system that may be utilized to execute the software of an embodiment of the invention.

FIG. 2 shows a system block diagram of the computer system of FIG. 1.

FIG. 3 shows how a Java source code program is executed.

FIG. 4 shows the components of an implementation of a Java runtime system.

FIG. 5 illustrates frames for functions that are stored on a Java stack.

FIG. 6 shows a process of storing information regarding active objects obtained during run-time.

FIG. 7 shows a process of presenting information regarding active objects obtained during run-time.

FIGS. 8A-8E show a hypertext document of all classes that had instances that were active when the snapshot was taken.

FIG. 9 shows a hypertext document of a class that had instances that were active when the snapshot was taken.

FIG. 10 shows a hypertext document of instances of a class excluding subclasses.

FIG. 11 shows a hypertext document of instances of a class including subclasses.

FIG. 12 shows a hypertext document of information regarding an active object.

FIG. 13 shows a hypertext document of root set references to an active object that excludes weak references.

FIG. 14 shows a hypertext document of root set references to an active object that includes weak references.

FIGS. 15A and 15B show a hypertext document of all objects reachable from an active object.

FIGS. 16A-16E show a hypertext document of all members of the root set.

FIG. 17 shows a process of determining differences (e.g., new instances) between active objects at two different run-times.

FIG. 18 shows a hypertext document of new instances of a class.

FIG. 19 shows a hypertext document of new instances of a class including subclasses.

FIGS. 20A-20B describe a data structure that may be utilized to store information regarding active objects.

FIG. 21 shows an embodiment of a BOD server.

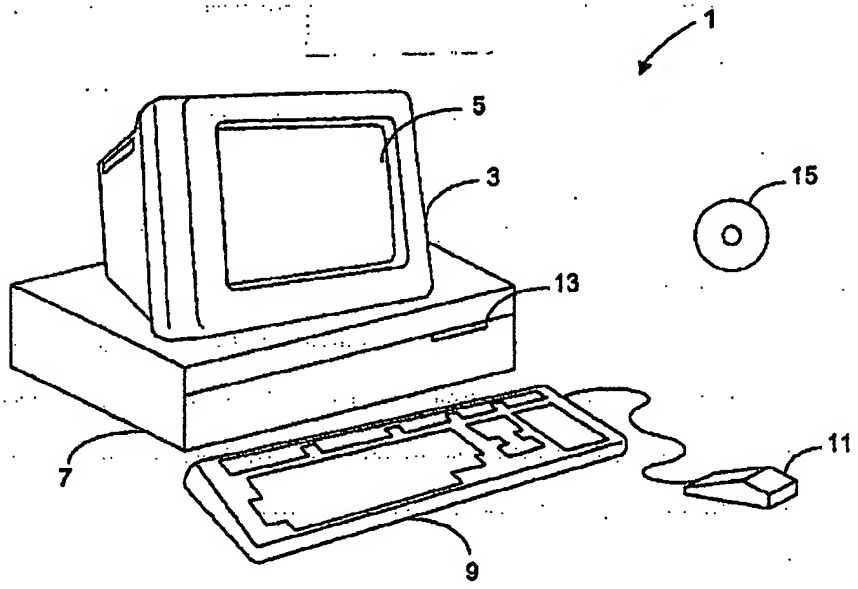


FIG. 1

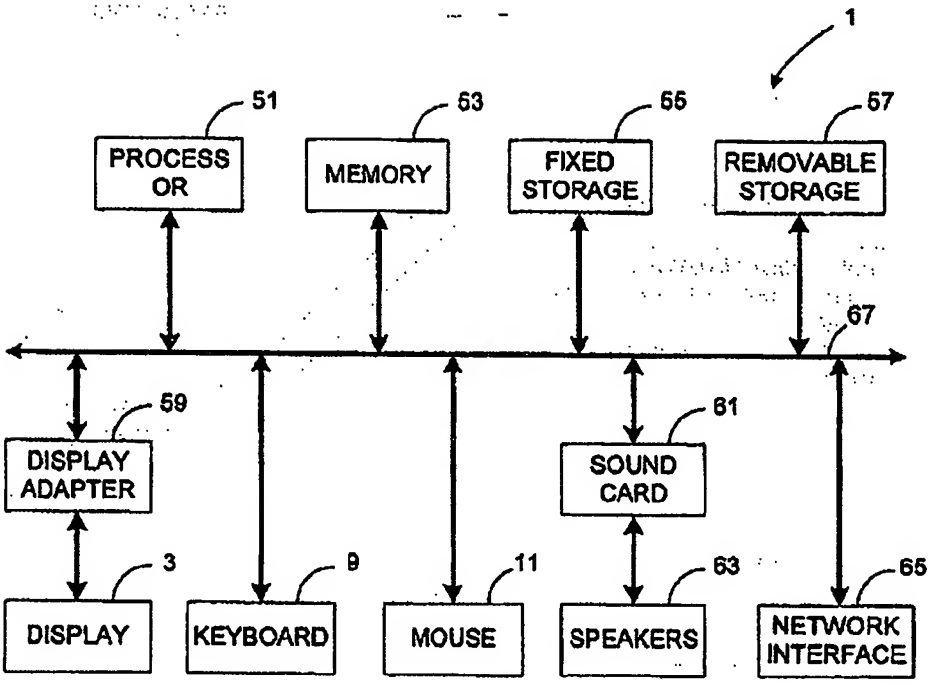

1
51
PROCESS OR
53
MEMORY
55
FIXED STORAGE
57
REMOVABLE STORAGE
67
59
DISPLAY ADAPTER
61
SOUND CARD
3
DISPLAY
9
KEYBOARD
11
MOUSE
63
SPEAKERS
65
NETWORK INTERFACE

FIG. 2

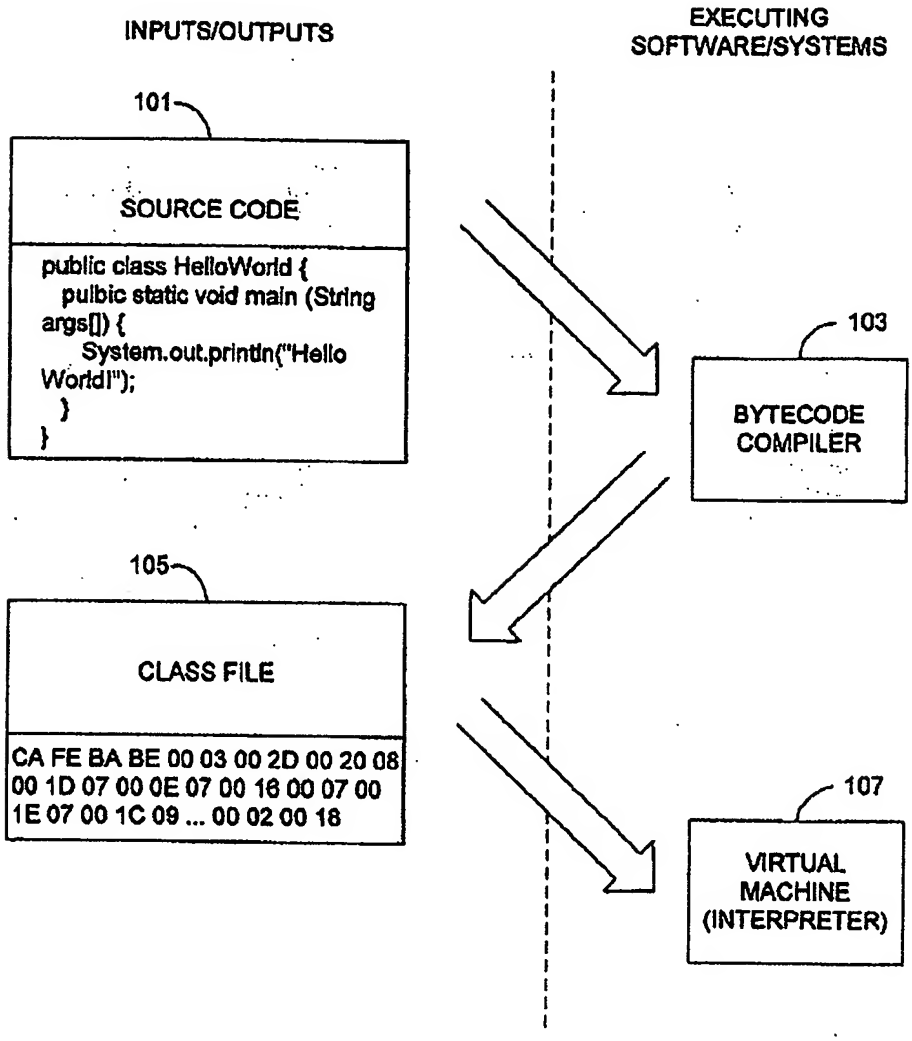

INPUTS/OUTPUTS
EXECUTING SOFTWARE/SYSTEMS
101
SOURCE CODE
public class HelloWorld {
pulbic static void main (String args[]) {
System.out.println("Hello World!");
}
}
103
BYTECODE COMPILER
105
CLASS FILE
CA FE BA BE 00 03 00 2D 00 20 08 00 1D 07 00 0E 07 00 16 00 07 00 1E 07 00 1C 09 ... 00 02 00 16
107
VIRTUAL MACHINE (INTERPRETER)

FIG. 3

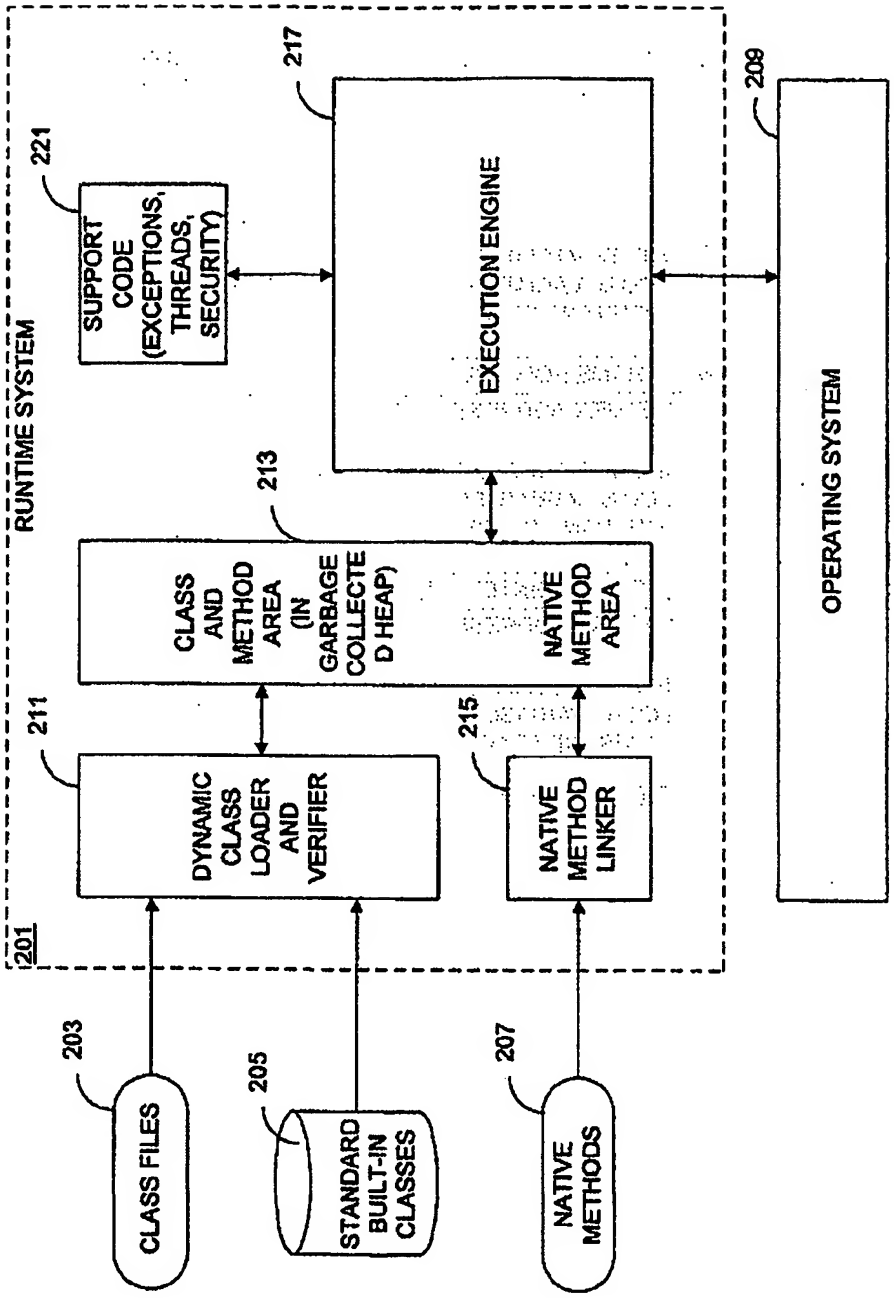

RUNTIME SYSTEM
201
SUPPORT CODE (EXCEPTIONS, THREADS, SECURITY)
221
EXECUTION ENGINE
217
OPERATING SYSTEM
209
CLASS AND METHOD AREA (IN GARBAGE COLLECTED HEAP)
NATIVE METHOD AREA
213
DYNAMIC CLASS LOADER AND VERIFIER
211
NATIVE METHOD LINKER
215
CLASS FILES
203
STANDARD BUILT-IN CLASSES
205
NATIVE METHODS
207

FIG. 4

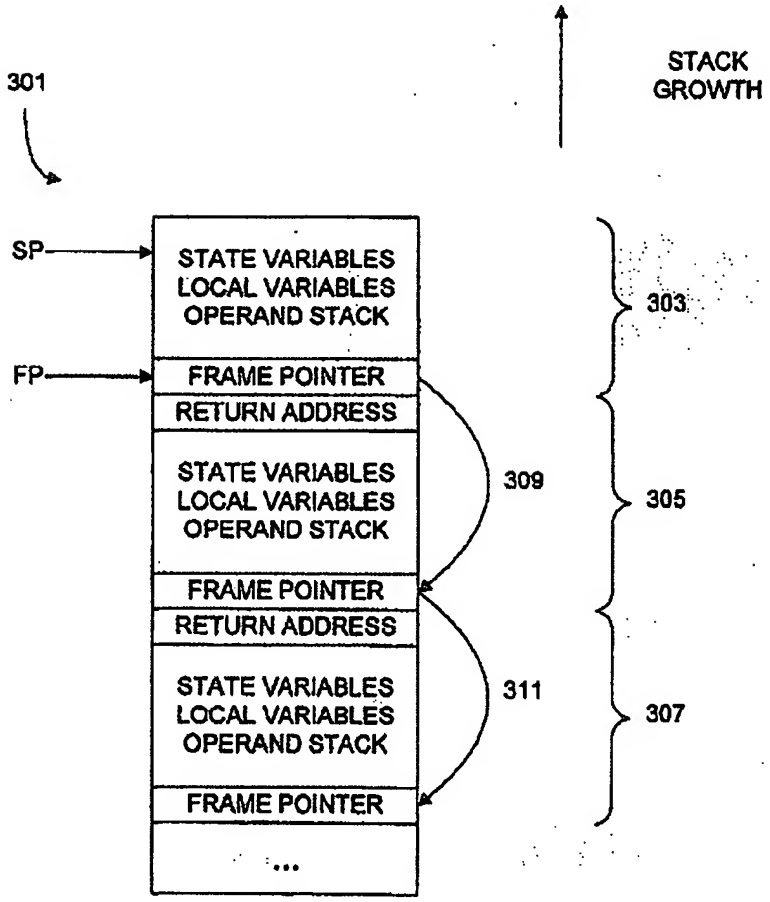

301
STACK GROWTH
SP
FP
STATE VARIABLES
LOCAL VARIABLES
OPERAND STACK
FRAME POINTER
RETURN ADDRESS
STATE VARIABLES
LOCAL VARIABLES
OPERAND STACK
FRAME POINTER
RETURN ADDRESS
STATE VARIABLES
LOCAL VARIABLES
OPERAND STACK
FRAME POINTER
...
303
305
307
309
311

FIG. 5

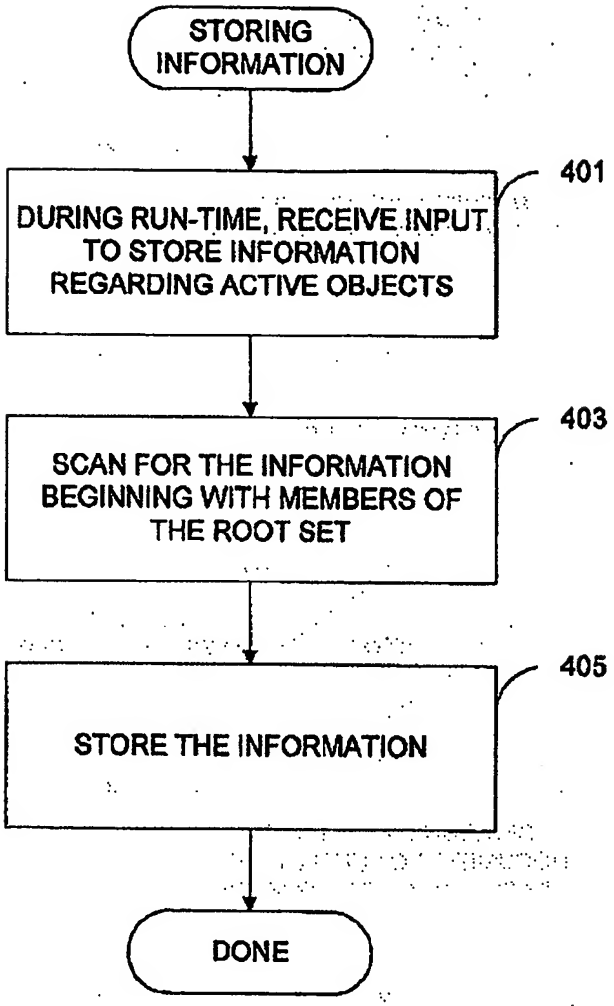

STORING INFORMATION
401
DURING RUN-TIME, RECEIVE INPUT TO STORE INFORMATION REGARDING ACTIVE OBJECTS
403
SCAN FOR THE INFORMATION BEGINNING WITH MEMBERS OF THE ROOT SET
405
STORE THE INFORMATION
DONE

FIG. 6

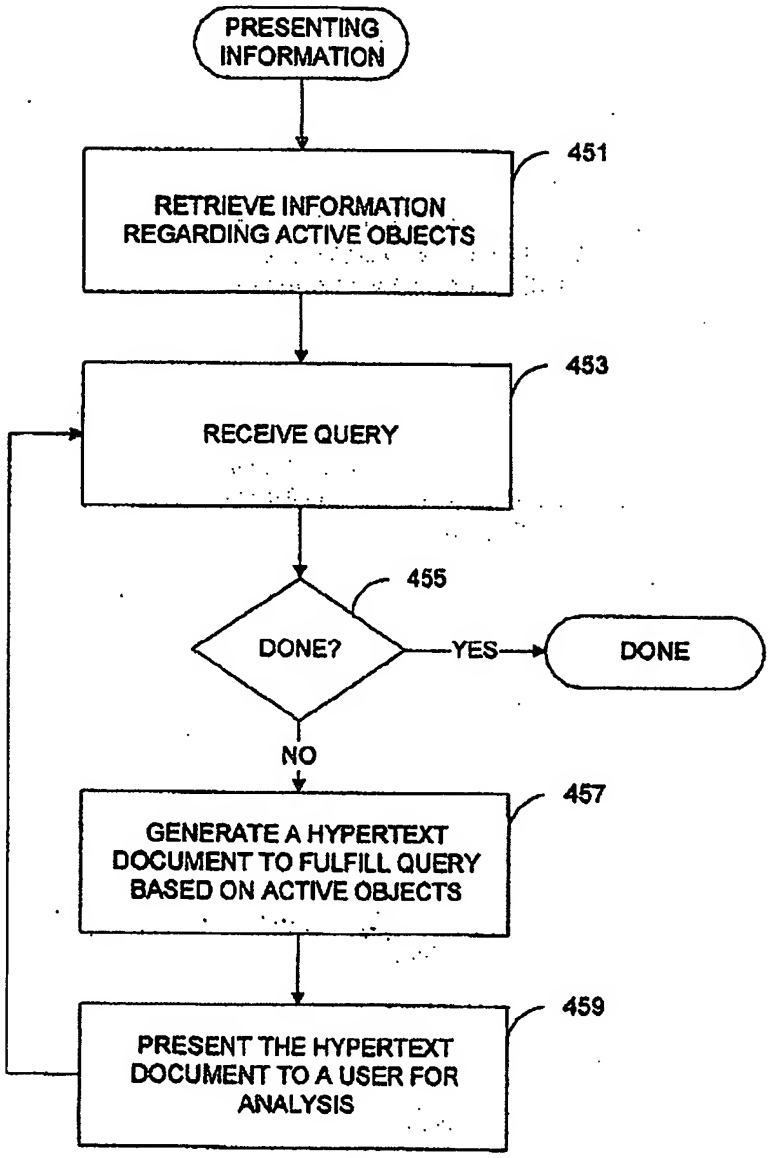

PRESENTING INFORMATION
451
RETRIEVE INFORMATION REGARDING ACTIVE OBJECTS
453
RECEIVE QUERY
455
DONE?
YES
DONE
NO
457
GENERATE A HYPERTEXT DOCUMENT TO FULFILL QUERY BASED ON ACTIVE OBJECTS
459
PRESENT THE HYPERTEXT DOCUMENT TO A USER FOR ANALYSIS

FIG. 7

ALL CLASSES

Package <Default Package>

class char
class double
class float
class int

Package java.applet

class java.applet.Applet

Package java.awt

class java.awt.AWTEvent
class java.awt.BorderLayout
class java.awt.Canvas
class java.awt.Color
class java.awt.Component
class java.awt.Container
class java.awt.Cusor
class java.awt.Dimension
class java.awt.Event
class java.awt.EventDispatchThread
class java.awt.EventQueue
class java.awt.EventQueueItem
class java.awt.FlowLayout
class java.awt.FocusManager
class java.awt.Font
class java.awt.Frame
class java.awt.Graphics
class java.awt.GridBagConstraints
class java.awt.GridBagLayout
class java.awt.GridBagLayoutInfo
class java.awt.Image
class java.awt.Insets
class java.awt.LayoutManager
class java.awt.LayoutManger2
class java.awt.LightweightDispatcher
class java.awt.MenuContainer
class java.awt.Panel
class java.awt.Point
class java.awt.Polygon
class java.awt.Rectangle
class java.awt.ScrollPane
class java.awt.Shape
class java.awt.Toolkit

Fig. 8A

class java.awt.Window

Package java.awt.event

class java.awt.event.ComponentEvent
class java.awt.event.FocusEvent
class java.awt.event.InputEvent
class java.awt.event.KeyEvent
class java.awt.event.MouseEvent
class java.awt.event.PaintEvent
class java.awt.event.WindowEvent

Package java.awt.image

class java.awt.image.ImageConsumer
class java.awt.image.ImageObserver
class java.awt.image.ImageProducer

Package java.awt.peer

class java.awt.peer.CanvasPeer
class java.awt.peer.ComponentPeer
class java.awt.peer.ContainerPeer
class java.awt.peer.FontPeer
class java.awt.peer.FramePeer
class java.awt.peer.LightweightPeer
class java.awt.peer.PanelPeer
class java.awt.peer.WindowPeer

Fig. 8B

Package java.io

class java.io.BufferedInputStream
class java.io.BufferedWriter
class java.io.File
class java.io.FileDescriptor
class java.io.FileInputStream
class java.io.FileOutputStream
class java.io.FilterInputStream
class java.io.FilterOutputStream
class java.io.InputStream
class java.io.OutputStream
class java.io.OutputStreamWriter
class java.io.PrintStream
class java.io.Serializable
class java.io.Writer

Package java.lang

class java.lang.Character
class java.lang.Class
class java.lang.Cloneable
class java.lang.Double
class java.lang.Float
class java.lang.Integer
class java.lang.Math
class java.lang.Number
class java.lang.Object
class java.lang.Runnable
class java.lang.Runtime
class java.lang.String
class java.lang.StringBuffer
class java.lang.System
class java.lang.System$DelegatingInputStream
class java.lang.System$DelegatingPrintStream
class java.lang.Thread
class java.lang.ThreadGroup

Package java.util

class java.util.Dictionary
class java.util.EventObject
class java.util.Hashtable
class java.util.HashtableEntry
class java.util.Locale
class java.util.Properties
class java.util.Random
class java.util.Vector

Fig. 8C

Package jovial.slotCar

class jovial.slotCar.Car
class jovial.slotCar.GasPedal
class jovial.slotCar.RaceApplet
class jovial.slotCar.RaceView
class jovial.slotCar.RandomGasPedal

Package jovial.slotCar.animator

class jovial.slotCar.animator.Animatee
class jovial.slotCar.animator.Animator
class jovial.slotCar.animator.Drawable

Package jovial.slotCar.track

class jovial.slotCar.track.CurvedTrackSegment
class jovial.slotCar.track.FigureEightTrack
class jovial.slotCar.track.StraightTrackSegment
class jovial.slotCar.track.Track
class jovial.slotCar.track.TrackPosition
class jovial.slotCar.track.TrackSegment

503

Package sun.awt

class sun.awt.AWTFinalizeable
class sun.awt.AWTFinalizer
class sun.awt.CharToByteSymbol
class sun.awt.DrawingSurface
class sun.awt.EmbeddedFrame
class sun.awt.FontDescriptor
class sun.awt.PlatformFont
class sun.awt.ScreenUpdater
class sun.awt.ScreenUpdaterEntry
class sun.awt.SunToolkit
class sun.awt.UpdateClient

Package sun.awt.image

class sun.awt.image.Image
class sun.awt.image.ImageFetchable
class sun.awt.image.ImageRepresentation
class sun.awt.image.ImageWatched
class sun.awt.image.InputStreamImageSource
class sun.awt.image.OffScreenImageSource

Package sun.awt.motif

class sun.awt.motif.CharToByteX11Dingbats
class sun.awt.motif.InputThread
class sun.awt.motif.MCanvasPeer
class sun.awt.motif.MComponentPeer
class sun.awt.motif.MEmbeddedFrame
class sun.awt.motif.MFontPeer
class sun.awt.motif.MFramePeer
class sun.awt.motif.MPanelPeer
class sun.awt.motif.MToolkit
class sun.awt.motif.X11Graphics
class sun.awt.motif.X11Image
class sun.awt.motif.X11OffScreenImage

Fig. 8D

Package sun.io

class sun.io.CharToByte8859_1
class sun.io.CharToByteConverter
class sun.io.CharacterEncoding

Package <Arrays>
class [I

Other Queries

705

- Show all members of the rootset

FIG. 8E

Class jovial.slotCar.track.TrackSegment

class jovial.slotCar.track.TrackSegment

Superclass:

class jovial.slotCar.animator.Drawable

Subclasses:

class jovial.slotCar.track.CurvedTrackSegment
class jovial.slotCar.track.StraightTrackSegment

Instance Data Members:

color_(Ljava/awt/Color;)
height_(I)
next_(Ljovial/slotCar/track/TrackSegment;)

Static Data Members:

Instances

553

Exclude subclasses
Include subclasses

575

Fig. 9

Instances of jovial.slotCar.Car

class jovial.slotCar.Car

jovial.slotCar.Car@0xee704610 (24bytes)
jovial.slotCar.Car@0xee7043e0 (24bytes)

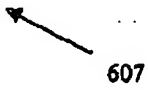


Fig. 10

Instances of jovial.slotCar.Car (including subclasses)

class jovial.slotCar.Car

jovial.slotCar.Car@0xee704610 (24bytes)
jovial.slotCar.Car@0xee7043e0 (24 bytes)

Fig. 11

Object at 0xee704610

instance of jovial.slotCar.Car (24 bytes)

Class:

class jovial.slotCar.Car

Instance data members:

drawColor_(Ljava/awt/Color;) :java.awt.Color@0xee703fa8 (12 bytes)
eraseColor_(Ljava/awt/Color;) :java.awt.Color@0xee703fc8 (12bytes)
gasPedal_(Ljovial/slotCar/GasPedal;) :jovial.slotCar.RandomGasPedal@0xee704608 (20 bytes)
poly_(Ljava/awt/Polygon;) :java.awt.Polygon@0xee7045f0 (20 bytes)
pos_(Ljovial/slotCar/track/TrackPosition;) :jovial.slotCar.track.TrackPosition@0xee704620 (32 bytes)

References to this object:

array@0xee704408 (12 bytes) : Element 1 of array
array@0xee704010 (44 bytes) : Element 7 of array

Other Queries

Reference Chains from Rootset

- Exclude weak refs
- Include weak refs

Objects reachable from here

635
675
681

Fig. 12

Rootset references to jovial.slotCar.Car
(excludes weak refs)

References to jovial.slotCar.Car@0xee704610 (24 bytes)

Java Local References

Java stack local (from java.lang.Thread):

→ jovial.slotCar.animator.Animator@0xee704058 (120 bytes) (field animatee_:)
→ jovial.slotCar.RaceView@0xee703ff8 (8bytes) (field cars_:)
→ array@0xee704408 (12 bytes) (Element 1 of array:)
→ jovial.slotCar.Car@0xee704610 (24 bytes)

Other queries

Include weak refs

691

Fig. 13

Rootset references to jovial.slotCar.Car
(includes weak refs)

References to jovial.slotCar.Car@0xee704610 (24 bytes)

Java Local References

Java stack local (from java.lang.Thread):

→ jovial.slotCar.animator.Animator@0xee704058 (120 bytes) (field animatee_:)
→ jovial.slotCar.RaceView@0xee703ff8 (8 btyes) (field cars_:)
→ array@0xee704408 (12 btyes) (Element 1 of array:)
→ jovial.slotCar.Car@0xee704610 (24 bytes)

Other queries

Exclude weak refs

693

Fig. 14

Objects Reachable From 0xee704610

jovial.slotCar.Car@0xee704610 (24 bytes)

jovial.slotCar.track.StraightTrackSegment@0xee704398 (88 bytes)
jovial.slotCar.track.StraightTrackSegment@0xee7041b8 (88 bytes)
jovial.slotCar.track.StraightTrackSegment@0xee7044d0 (88 bytes)
jovial.slotCar.track.StraightTrackSegment@0xee7043b8 (88 bytes)
jovial.slotCar.track.CurvedTrackSegment@0xee7043a0 (60 bytes)
jovial.slotCar.track.CurvedTrackSegment@0xee7041c8 (60 bytes)
jovial.slotCar.track.CurvedTrackSegment@0xee704418 (60 bytes)
jovial.slotCar.track.TrackPosition@0xee704620 (32 bytes)
<array>@0xee704138 (20 bytes)
<array>@0xee704130 (20 bytes)
<array>@0xee704460 (20 bytes)
<array>@0xee704358 (20 bytes)
<array>@0xee7045d0 (20 bytes)
<array>@0xee7043b0 (20 bytes)
<array>@0xee7044d8 (20 bytes)
<array>@0xee7045d8 (20 bytes)
<array>@0xee7042e8 (20 bytes)
<array>@0xee7043a8 (20 bytes)
<array>@0xee704420 (20 bytes)
<array>@0xee704168 (20 bytes)
<array>@0xee7044e0 (20 bytes)
<array>@0xee704268 (20 bytes)
<array>@0xee7042a8 (20 bytes)
<array>@0xee704170 (20 bytes)
<array>@0xee7042a0 (20 bytes)
<array>@0xee704360 (20 bytes)
<array>@0xee704490 (20 bytes)
<array>@0xee7042f0 (20 bytes)
<array>@0xee704320 (20 bytes)
<array>@0xee704328 (20 bytes)
<array>@0xee704498 (20 bytes)
<array>@0xee704510 (20 bytes)
<array>@0xee704428 (20 bytes)
<array>@0xee704270 (20 bytes)
java.awt.Polygon@0xee704288 (20 bytes)
java.awt.Polygon@0xee704308 (20 bytes)
java.awt.Polygon@0xee7042c0 (20 bytes)
java.awt.Polygon@0xee704340 (20 bytes)
java.awt.Polygon@0xee704260 (20 bytes)
java.awt.Polygon@0xee704188 (20 bytes)
java.awt.Polygon@0xee7044b0 (20 bytes)
java.awt.Polygon@0xee7045f0 (20 bytes)

Fig. 15A

java.awt.Polygon@0xee7044f8 (20 bytes)
java.awt.Polygon@0xee704440 (20 bytes)
java.awt.Polygon@0xee704118 (20 bytes)
java.awt.Polygon@0xee704378 (20 bytes)
java.awt.Polygon@0xee704478 (20 bytes)
jovial.slotCar.RandomGasPedal@0xee704608 (20 bytes)
array@0xee704390 (12 bytes)
array@0xee7041a0 (12 bytes)
array@0xee7042d8 (12 bytes)
array@0xee7044c8 (12 bytes)
java.awt.Color@0xee703fa8 (12 bytes)
java.awt.Color@0xee703fc8 (12 bytes)
java.awt.Point@0xee706e58 (12 bytes)

Total size: 1472 bytes.

Fig. 15B

All Members of the Rootset

Java Static References

Static reference from java.awt.BorderLayout.CENTER:
→ java.lang.String@0xee703e58 (16 bytes)
Static reference from java.awt.BorderLayout.BAST:
→ java.lang.String@0xee703e78 (16 bytes)
Static reference from java.awt.BorderLayout.NORTH:
→ java.lang.String@0xee703e98 (16 bytes)
Static reference from java.awt.BorderLayout.SOUTH:
→ java.lang.String@0xee703e88 (16 bytes)
Static reference from java.awt.BorderLayout.WEST:
→ java.lang.String@0xee703e68 (16 bytes)
Static reference from java.awt.Canvas.base:
→ java.lang.String@0xee704060 (16 bytes)
Static reference from java.awt.Color.black:
→ java.awt.Color@0xee703fc8 (12 bytes)
Static reference from java.awt.Color.blue:
→ java.awt.Color@0xee704240 (12 bytes)
Static reference from java.awt.Color.cyan:
→ java.awt.Color@0xee704248 (12 bytes)
Static reference from java.awt.Color.darkGray:
→ java.awt.Color@0xee703fd0 (12 bytes)
Static reference from java.awt.Color.gray:
→ java.awt.Color@0xee703fd8 (12 bytes)
Static reference from java.awt.Color.green:
→ java.awt.Color@0xee703fa0 (12 bytes)
Static reference from java.awt.Color.lightGray:
→ java.awt.Color@0xee703fe0 (12 bytes)
Static reference from java.awt.Color.magenta:
→ java.awt.Color@0xee704250 (12 bytes)
Static reference from java.awt.Color.orange:
→ java.awt.Color@0xee703fb0 (12 bytes)
Static reference from java.awt.Color.pink:
→ java.awt.Color@0xee703fb8 (12 bytes)
Static reference from java.awt.Color.red:
→ java.awt.Color@0xee703fc0 (12 bytes)
Static reference from java.awt.Color.white:
→ java.awt.Color@0xee703fe8 (12 bytes)
Static reference from java.awt.Color.yellow:
→ java.awt.Color@0xee703fa8 (12 bytes)
Static reference from java.awt.Component.LOCK
→ java.lang.Object@0xee703d20 (4bytes)
Static reference from java.awt.Component.actionListenerK:
→ java.lang.String@0xee703dd8 (16 bytes)

FIG. 16A

Static reference from java.awt.Component.adjustmentListenerK:
→ java.lang.String@0xee703dc8 (16 bytes)
Static reference from java.awt.Component.componentListenerK:

•
•
•

Static reference from sun.io.CharToByteConverter.pkgString:
→ java.lang.String@0xee700538 (16 bytes)
Static reference from sun.io.CharacterEncoding.aliasTable:
→ java.util.Hashtable@0xee7005c8 (20 bytes)

Java Local References

Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.X11Graphics@0xee707a60 (36 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventDispatchThread@0xee7052e8 (56 bytes)
Java stack local (from java.lang.Thread):
→ jovial.slotCar.animator.Animator@0xee704058 (120 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.MToolkit@0xee705368 (8 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ java.lang.ThreadGroup@0xee703ba8 (44 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.event.FocusEvent@0xee707648 (24 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ jovial.slotCar.animator.Animator@0xee704058 (120 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.MToolkit@0xee705368 (8 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.X11Graphics@0xee707a48 (36 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Java stack local (from java.awt.EventDispatchThread):
→ java.awt.event.FocusEvent@0xee707648 (24 bytes)
Java stack local (from java.lang.Thread):

FIG. 16B

→ sun.awt.motif.MCanvasPeer@0xee707580 (16 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707e08 (48 bytes)
Java stack local (from java.lang.Thread):
→ java.lang.Thread@0xee7052b0 (48 bytes)
Java stack local (from java.lang.Thread):
→ sun.awt.motif.X11OffScreenImage@0xee707ad0 (32 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707e08 (48 bytes)
Java stack local (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707e08 (48 bytes)
Java stack local (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalzier@0xee707e08 (48 bytes)
Java stack local (from sun.awt.ScreenUpdater):
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)

Native Static References

Native code reference:
→ sun.awt.AWTFinalizer@0xee707e08 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee7000b0 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee703e20 (48 bytes)
Native code reference:
→ java.awt.EventDispatchThread@0xee7052e8 (56 bytes)
Native code reference:
→ sun.awt.ScreenUpdater@0xee707c28 (52 bytes)
Native code reference:
→ java.lang.Thread@0xee703c48 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee700088 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference:
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee703a08 (48 bytes)
Native code reference:
→ java.lang.Thread@0xee7052b0 (48 bytes)

FIG. 16C

Native Local References

Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Native code reference(from java.lang.Thread):
→ sun.awt.motif.MToolkit@0xee705368 (8bytes)
Native code reference(from java.lang.Thread):
→ java.lang.Thread@0xee703c48 (48 bytes)
Native code reference (from java.lang.Thread):
→ array@0xee703f88 (4 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Native code reference (from java.lang.Thread):
→ array@0xee703f88 (4 bytes)
Native code reference (from sun.awt.motif.InputThread):

•
•
•

→ sun.awt.AWTFinalizer@0xee707e08 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventDispatchThread@0xee7052e8 (56 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.EventQueue@0xee705338 (8 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.Event@0xee707078 (56 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from java.awt.EventDispatchThread):
→ java.awt.Event@0xee7075c8 (56 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7052b0 (48 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from sun.awt.motif.InputThread):
→ sun.awt.motif.InputThread@0xee7052d0 (48 bytes)
Native code reference (from java.lang.Thread):

FIG. 16D

→ java.lang.Thread@0xee7052b0 (48 bytes)
Native code reference (from sun.awt.AWTFinalizer):
→ sun.awt.AWTFinalizer@0xee707e08 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703c90 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.awt.Point@0xee706e08 (12 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7052b0 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee7077c8 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703a08 (48 bytes)
Native code reference (from java.lang.Thread):
→ java.lang.Thread@0xee703a08 (48 bytes)

Other Queries

- Show All Classes

715

FIG. 16E

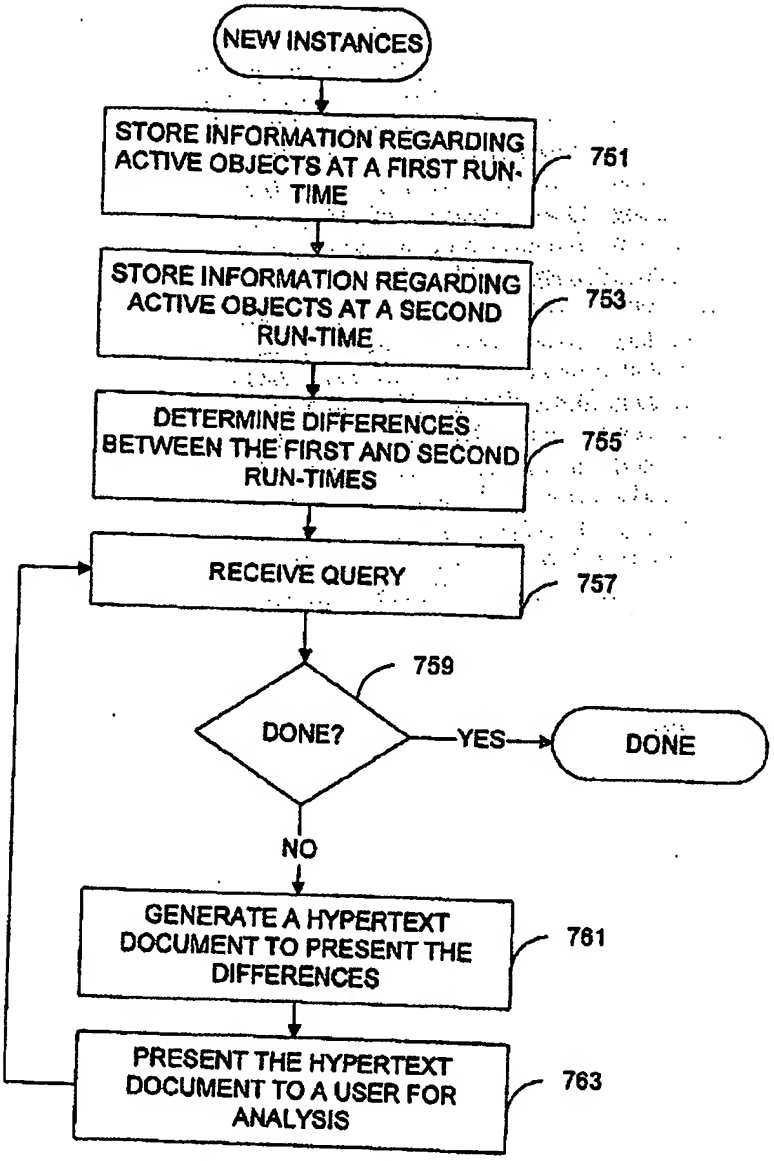


FIG. 17

New instances of java.awt.Point

class java.awt.Point ← 801

java.awt.Point@0xee706e58[new] (12 bytes)
java.awt.Point@0xee706e48[new] (12 bytes)
java.awt.Point@0xee706e08[new] (12 bytes)
803

FIG. 18

New instances of java.lang.Object (including subclasses)

class java.lang.Object

java.awt.Point@0xee706e58[new] (12 bytes)
java.awt.Point@0xee706e48[new] (12 bytes)
java.awt.Point@0xee706e08[new] (12 bytes)

FIG. 19

BOD FILE STRUCTURE
@(#) bod_format.txt

This document gives the file format of a bod ("binary object dump") file. In this version, reported data types include int, char array, and reference – fields of other types simply aren't reported. This description is in an informal BNF-style syntax, where sometimes a symbol in a grammar is given by itself (i.e. "class"), and sometimes it is given following a descriptive name and a colon (i.e. statics:: static_field[]). The array syntax means "zero or more of this element".

A "thing" (like a class or an object) may appear in the dump file more than once. This rule is intended primarily to make the VM implementation that produces the file easier. If a thing appears twice, it is expected that it will have the same values, except, perhaps for the root_flags and the thread_id. The set of root of an object is the union of the roots specified by all occurrences of the object in the union of the roots specified by all occurrences of the object in the file (i.e. if an object is referred to by two local variables in two different threads, it will appear at least twice in the file, with an entry indicating each local variable root appearing at least once).

bod file ::= magic_number version_number thing[]

magic_number ::= 0x0b0dd00d

version_number ::= 4

thing ::= type:byte id:int root_flags:byte thread_id:int stuff

type is object=0, class=1, object array=2, char array=3, other array=4
root_flags is static=1, java stack local=2, native ref=4
thread_id is id field of the thread object responsible for this being a root, or 0 if not applicable

stuff ::= object | class | object_array | char_array

object ::= class_id : int
num_fields :int
data : int[]

data is the value of int fields, or the id of object/array fields

FIG. 20A

```
class ::= superclass_id : int
          name_len : int
          num_statics : int
          statics : static_field[]
          num_fields : int
          fields : field[]
          instance_size : int                    // in bytes, including VM overhead

field ::= name_len : int
          name : char[]
          signature_len : int
          signature : char[]

static_field ::= field
                 value : int                     // value is value of int field, or id of object

object_array ::= size : int                      // in bytes, including VM overhead
                 len : int
                 object_id : int[]

char_array ::= size : int
               len : int
               data : char[]

Other_array ::= size : int
```

FIG. 20B

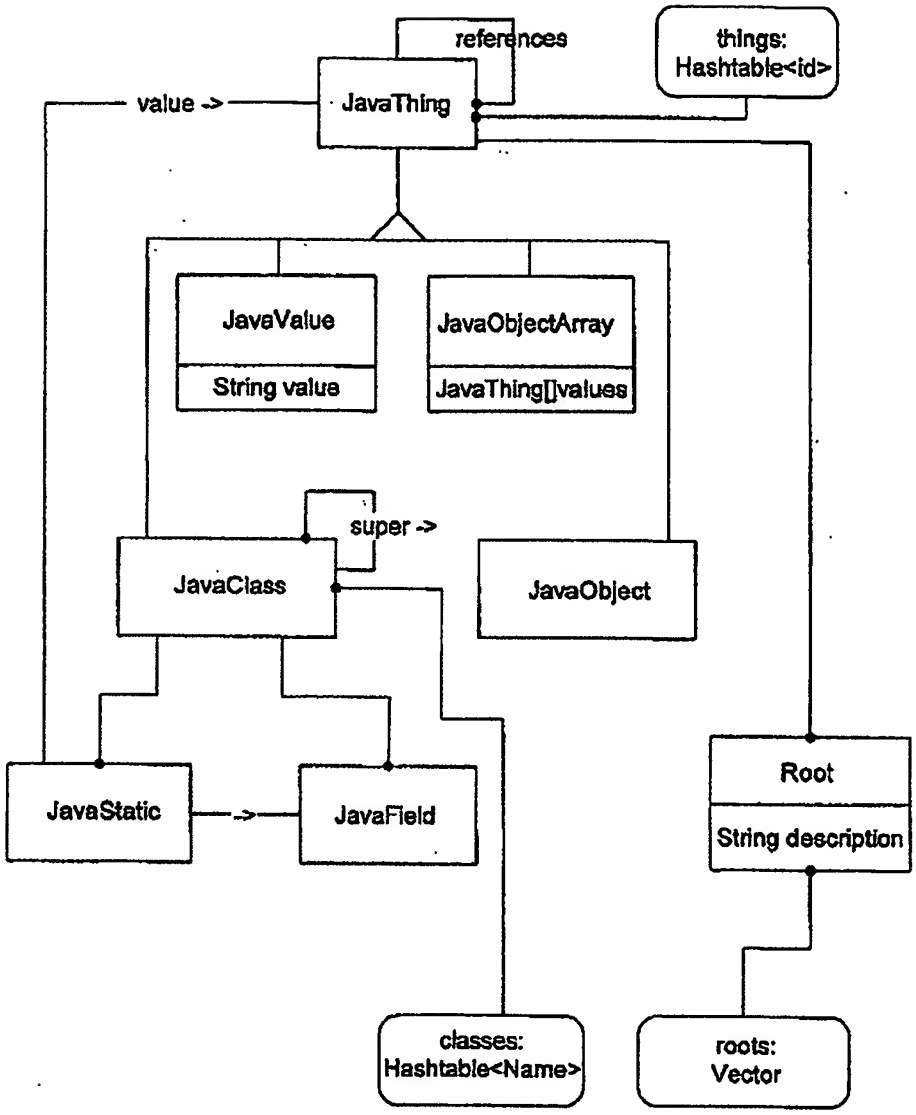


FIG. 21

[Abstract

Techniques for analyzing object-oriented computer programs are provided. A snapshot of the objects that are active at a specific point in time during execution may be stored. An analysis tool may be utilized to generate hypertext documents that allow a user to analyze the active objects. Additionally, a user may compare two different snapshots of active objects at two different run-times so that, for example, new instances of a class may be easily identified.